Proyecto fin de carrera

Ingeniería Técnica Mecánica

SOFTWARE PARA MÁQUINA DE MEDICIÓN POR COORDENADAS DIDÁCTICA

Autor: José Tovío Ciércoles

Director del proyecto: Mª Rosario González Pedraza

Especialidad: Ingeniería Técnica Mecánica

Fecha: 21 de mayo de 2015

Convocatoria: junio de 2015

Resumen

El presente documento recoge el resultado del proyecto de fin de carrera que ha consistido en la implementación de un programa llamado *Descartes* para una máquina de medición por coordenadas.

Una máquina de medición por coordenadas es un instrumento de medida capaz de determinar las medidas y la forma del objeto sobre el que se aplique. Para ello dispone de un palpador que se posa sobre la superficie de la pieza a medir. El palpador se mueve mediante tres ejes perpendiculares y cuando está parado, estos ejes proporcionan las coordenadas cartesianas X, Y, Z del centro del palpador.

El programa utiliza la nube de puntos proporcionada por el palpador y determina los parámetros de la superficie palpada (esfera, plano, cilindro o cono). También calcula distancias, ángulos e intersecciones entre las superficies medidas.

El programa está implementado en *Lazarus* que es un entorno de programación visual basado en Pascal y es software libre. Es fácil de utilizar y permite añadir nuevas funcionalidades.

En las pruebas realizadas los resultados obtenidos han sido totalmente satisfactorios y demuestran que el programa supera las capacidades de la máquina original.

Índice de contenido

1. Introducción

En esta memoria se presenta un programa para una máquina de medición por coordenadas. La máquina de medición por coordenadas utilizada está conectada a un terminal con capacidades de almacenamiento y de cálculo bastante limitadas. El programa desarrollado amplía las posibilidades de la máquina ya que permite obtener los parámetros de más objetos además de permitir el uso de un número mayor de puntos para determinar los parámetros de los objetos que ya calculaba la máquina. Asimismo el programa está implementado en lenguaje Pascal orientado a objetos, en concreto en la versión del entorno Lazarus, que es software libre disponible para Windows, Mac y Linux. De este modo las opciones de ampliación o modificación del programa son mayores.

1.1. Descripción de la máquina

Una máquina de medición por coordenadas es un instrumento de medida capaz de determinar las medidas y la forma del objeto sobre el que se aplique. Para ello dispone de un palpador que se posa sobre la superficie de la pieza a medir. El palpador se mueve mediante tres ejes perpendiculares y cuando está parado, estos ejes proporcionan las coordenadas cartesianas X, Y, Z del centro del palpador. Tomando una muestra de estas medidas (supondremos que no más de 20) e indicando el tipo de superficie que está siendo medida, la máquina determina sus principales parámetros.
La máquina en la que se ha basado el presente proyecto es una MXF203 que está conectada a una MICROPAK 100, ambas del la empresa Mitutoyo tal y como aparece en las ilustraciones 1 y 2.

Ilustración 1: Máquina MXF203

El resultado de las mediciones de la máquina se muestra en la pantalla del MICROPAK 100 y también pude imprimirse en papel.

Sin embargo, como es fácil adivinar, la máquina MICROPAK 100 tiene una serie de limitaciones, además de ser poco práctica a la hora de almacenar y manipular los datos que se obtienen del palpador. Entre las limitaciones destacamos:

- Dificultad de uso, siendo este poco intuitivo.
- Capacidad muy pobre para manipular los datos que se obtienen del palpador.
- No hay opciones de ampliar el tipo de funciones que realiza.
- No realiza medidas de conos de ningún tipo.
- No realiza medidas de cilindros cuyo eje no sea perpendicular al plano base.
- No muestra información del error cometido en el ajuste.

En definitiva, cualquier ordenador conectado al palpador con un programa específico es capaz de realizar muchas más funciones y de una forma más amigable para el usuario.

Ilustración 2: Micropak 100

2. Objetivos

El principal objetivo del trabajo ha consistido en desarrollar e implementar un programa que procese los datos obtenidos por una máquina de medición por coordenadas. El programa debe tener una interfaz amigable para el usuario y fácil de manejar. Además debe ser capaz de realizar todas las funciones que ya posee el Micropak 100 y si es posible debe realizar otras funciones adicionales. Los datos introducidos deben poder guardarse en archivos así como las listas de objetos manejadas. También es necesario que se pueda interactuar con estos objetos: calcular distancias, ángulos, etc.
El programa debe implementarse en un entorno de programación que sea software libre y que pueda portarse a otras plataformas.
De forma esquemática podemos resumir los objetivos que tiene que cumplir el programa en la siguiente lista:

- Interfaz de ventanas sencilla de manejar.
- Posibilidad de exportar datos o de introducirlos directamente.
- Posibilidad de guardar datos.
- Posibilidad de guardar y cargar listas de objetos.
- Capacidad para realizar cálculos con los objetos.
- Programa multiplataforma.
- Desarrollado con herramientas de software libre.
- Opciones de ampliación.
- Que realice las mismas funciones que el Micropak 100.
- Que realice nuevas funciones.
- Opción de que pueda trabajar integrado con otras herramientas de software.

3. Alcance

En primer lugar el programa calcula las superficies definidas por el palpador: planos, cilindros, conos y esferas. Recibe como entrada una lista de puntos (20 como máximo) y calcula los parámetros que definen la superficie correspondiente. El usuario del programa debe cargar la lista de puntos, indicar la posición del palpador respecto del objeto e indicar el tipo de objeto.

Los elementos geométricos se almacenan como una fila de una matriz que consta de los parámetros que los definen. Para ello se han considerado los parámetros que se indican a continuación.

Superficies determinadas directamente por el palpador:

- **Plano**: 6 parámetros: un punto del plano *(a,b,c)* y un vector normal unitario *(u,v,w)*.
- **Esfera**: 4 parámetros: el centro *(a,b,c)* y el radio *r*.
- **Cilindro recto**: 3 parámetros: el centro *(a,b)* y el radio *r*.
- **Cono recto**: 4 parámetros: el vértice *(a,b,c)* y el parámetro *r* en la ecuación $(x-a)^2+(y-b)^2=r(z-c)^2$. Geométricamente este parámetro es el cuadrado de la tangente del semiángulo ($\tan a=\sqrt{r}$).
- **Cilindro girado**: 7 parámetros: un punto del eje *(a,b,c*), un vector dirección *(u,v,w*) y el radio *r*.
- **Cono girado**: 7 parámetros: un punto del eje *(a,b,c)*, un vector dirección *(u,v,w)* y el cuadrado de la tangente del semiángulo.

Observemos que no se ha dado el menor número de parámetros necesarios, sino que se dan más parámetros que hacen que la representación sea más sencilla y que sobre todo hacen más fáciles los cálculos posteriores derivados.

Elementos derivados mediante cálculos:

- **Punto**: 3 parámetros: el propio punto *(a,b,c)*.
- **Recta**: 6 parámetros: un punto *(a,b,c)* y un vector dirección *(u,v,w)*.

En segundo lugar, el programa tiene que calcular intersecciones, ángulos y distancias entre los distintos objetos fundamentales que maneja.

Por último, el programa tiene que permitir guardar una lista de objetos para poder recuperarla en una sesión posterior.

El programa Descartes supera ampliamente las posibilidades del Micropak 100 conectado originalmente a la máquina ya que proporciona una serie de funciones que no están disponibles en el Micropak 100:

- Permite introducir una nube de puntos y muestra el error máximo cometido, en lugar de limitarse a unos puntos fijos.
- Determina los parámetros de un cono.
- Determina los parámetros de un cono girado.
- Determina los parámetros de un cilindro girado.
- Permite guardas nubes de puntos y recuperarlas en distintas sesiones.
- Permite guardar y recuperar listas de objetos.
- Calcula distancias y ángulos entre todos los objetos guardados.

4. Metodología

Para poder lograr los objetivos marcados, en primer lugar resulta imprescindible comprender bien el problema. Para ello he tenido que repasar los conceptos geométricos y algebraicos necesarios y he utilizado principalmente los libros que cito en la bibliografía.

Una vez comprendido el problema, he investigado por internet tratando de encontrar trabajos similares o bien los algoritmos necesarios, sin embargo no he encontrado gran cosa y lo que he encontrado no se aplicaba directamente a los problemas que tenía planteados.

En una primera fase trabajé con el lenguaje de programación Pascal, por ser el que mejor conocía, pero enseguida pasé a utilizar Lazarus. Los pequeños problemas de programación y las dudas que me iban surgiendo con el entorno las iba resolviendo a medida que iban apareciendo.

Una vez que obtuve una primera versión funcional del programa, observe que la interfaz de usuario podía simplificarse y mejorarse. La versión definitiva del programa presenta ya está interfaz simplificada.

El ajuste por mínimos cuadrados para el plano, el cilindro y la esfera, aunque me supuso un esfuerzo considerable, considero que lo he resuelto de forma plenamente satisfactoria.

Sin embargo, con el cono, el cilindro girado y el cono girado he tenido más dificultades. El ajuste por mínimos cuadrados para el cono recto no funcionaba y me llevó bastante tiempo desarrollar otras estrategias. Para las figuras giradas no veía una estrategia clara en un principio y tras varios intentos infructuosos aplicando el método de mínimos cuadrados me decidí por utilizar un plano auxiliar para reducir estos casos al caso vertical.

Una vez conseguida una versión funcional del programa, procedí a comprobarlo con datos experimentales obtenidos de la propia máquina, ya que hasta entonces utilizaba datos obtenidos por simulación. En esta etapa tuve que corregir algunos errores que se pusieron de manifiesto. Los resultados que aparecen al final de la memoria recogen las pruebas realizadas con los datos experimentales y como puede verse en esa parte de la memoria estos resultados son muy satisfactorios.

El resultado final del proyecto es el programa Descartes 1.0 que espero que sea de utilidad para la máquina de medición por coordenadas.

4.1. Algoritmos de ajuste

Pueden considerarse varias posibilidades, aunque algunas de ellas se han descartado.

a) Cálculo exacto. Se palpan los puntos estrictamente necesarios para determinar los parámetros. Los errores de medición provocarían que los parámetros determinados no se correspondiesen con la superficie medida. Es por ello que es necesario tomar más puntos para realizar el cálculo de los parámetros.

b) Media de los cálculos exactos. Se palpan más puntos de los necesarios para definir la superficie y tomando todas las combinaciones se hace el cálculo exacto de los parámetros. Después se hace la media de estos parámetros ya calculados.

c) Mínimos cuadrados. Se palpan más puntos de los necesarios para definir la superficie. Se considera entonces que los parámetros de la superficie definen una familia de superficies y se calcula la distancia de cada uno de los puntos palpados a una superficie genérica de la familia. Se eleva al cuadrado dicha distancia y se calcula la suma para todos los puntos palpados. Los parámetros que mejor aproximan a la nube de puntos se obtienen al minimizar dicha función.

Los objetos introducidos se almacenan en una lista que posteriormente permite hacer cálculos de otros objetos y parámetros necesarios.

4.2. Programación: Entorno Lazarus

El programa se ha implementado en *Lazarus* que es un software libre basado en el lenguaje de programación *Pascal* y que puede considerarse un clon del *Delphi*. *Lazarus* está disponible para varios sistemas operativos (Windows, Mac, Linux), por lo que basta con compilar el código fuente en cada uno de ellos para obtener una aplicación ejecutable. *Lazarus* posee un entorno visual muy fácil de manejar y unas bibliotecas de objetos muy amplias.

Entre las razones que consideradas para elegir *Lázarus* para implementar este proyecto cabe mencionar las siguientes:

- **Sencillez y facilidad de uso**: el lenguaje está basado en *Pascal*, es por tanto un lenguaje fuertemente estructurado y fácil de aprender.
- **Posibilidad de desarrollar una aplicación con interfaz de ventanas**: *Lazarus* tiene todas las herramientas necesarias para desarrollar una aplicación con interfaz de ventanas sin tener que preocuparse excesivamente de dicho interfaz.
- **Multiplaforma**: como ya se ha dicho funciona en Windows, Linux y Mac, lo que hace que el programa pueda portarse a estas plataformas.
- **Software libre**: no es necesario comprar una licencia y por lo tanto cualquier persona puede realizar las mejoras que quiera si dispone del código fuente del programa, sin necesidad de ningún gasto.

4.3. Geometría y cálculo: Método de mínimos cuadrados

Para la parte matemática se han consultado diversos libros que aparecen en la bibliografía. En particular se han usado *Introducción al cálculo numérico* de Carlos Moreno González , *Álgebra y Geometría* de Eugenio Hernández y los libros *Cálculus I* y *Cálculus II* de Tom M. Apostol.

4.3.1 Sistemas lineales

El problema que se plantea puede resumirse de la siguiente manera: El palpador toma una muestra de puntos (limitados como máximo a 20) de cierta superficie. Además el usuario reconoce visualmente la forma de la superficie que quiere determinar (plano, esfera, cilindro o cono) y pulsa el botón correspondiente del programa. La tarea del programa consiste en ajustar dicha nube de puntos a la ecuación de la superficie que mejor la aproxima.

En varias ocasiones se llega al siguiente problema: dado un sistema de ecuaciones lineales $A\cdot\vec{x}=\vec{b}$, determinar la solución. Si hay más ecuaciones que incógnitas y los coeficientes de la matriz A proceden de una medición, el más mínimo error hará que el sistema sea incompatible. Observemos que en ese caso $A\cdot\vec{x}$ es una combinación lineal de las columnas de la matriz, y se buscan los coeficientes de dicha combinación lineal (es decir, el vector $\vec{x}$) para que el vector resultante esté a la menor distancia posible del vector $\vec{b}$. La solución geométrica está clara, $A\cdot\vec{x}$ debe ser la proyección ortogonal del vector $\vec{b}$ sobre el subespacio lineal determinado por las columnas de A . Por lo tanto, tiene que cumplirse que $\langle A\cdot\vec{y}\,,b-A\cdot\vec{x}\rangle=\vec{0}$ para todo vector $\vec{y}$. Así

$$A\cdot\vec{y}\,(b-A\cdot\vec{x})=\vec{0}\quad\forall\,\vec{y}\;\rightarrow\;A'(b-A\vec{x})=\vec{0}\;\rightarrow\;A'\vec{b}-A'A\vec{x}=\vec{0}\rightarrow A'b=A'A\vec{x}$$

Ecuación 1:
Sistema lineal

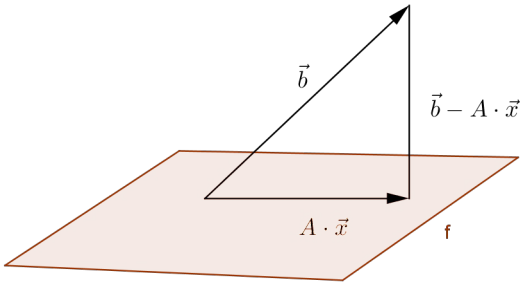


Ilustración 3: Mínimos cuadrados lineal

Este nuevo sistema tiene el mismo número de ecuaciones que de incógnitas y por lo tanto tendrá solución única si $A'A$ no es singular.

4.3.2 Ajustar superficies

Supongamos que tenemos una familia de superficies S_i con $i=(a,b,c,...)$ una serie de parámetros. Si se da una nube de n puntos $P_1=(x_1, y_1, z_1), ..., P_n=(x_n, y_n, z_n)$ hay que buscar el valor del parámetro i de modo que la superficie S_i es la que mejor se ajusta a esa nube de puntos. Para ello se calcula la distancia de cada punto a la superficie $d(P_k, S_i)$. Esta distancia depende de los parámetros i . Como habitualmente aparece una raíz cuadrada, en lugar de considerar la distancia se suele utilizar el cuadrado de la distancia. Si sumamos ahora todos los cuadrados obtenemos la función

$$F(i)=\sum_{k=1}^{n}(d(P_k, S_i))^2$$

Ecuación 2: Función de mínimos cuadrados

Si encontramos el valor de los parámetros i que hace que esta suma sea mínima, tendremos la superficie que mejor se ajusta a la nube de puntos que nos dan.

Por simplificar la notación se supone que tenemos únicamente tres parámetros (a,b,c) . Para minimizar la función $F(a,b,c)$ es necesario resolver el sistema

$$\left\{\begin{array}{l}\frac{\partial f}{\partial a}(a,b,c)=0\\ \frac{\partial f}{\partial b}(a,b,c)=0\\ \frac{\partial f}{\partial c}(a,b,c)=0\end{array}\right\}$$

Ecuación 3: Sistema de mínimos cuadrados

Este sistema no se puede resolver habitualmente de forma directa y es por ello que hay que recurrir a métodos numéricos. Para ello se necesita no solo el gradiente de F, sino también el hessiano de F, que se abrevian de la forma siguiente

$$\nabla F=\left(\begin{array}{c}\frac{\partial f}{\partial a}\\ \frac{\partial f}{\partial b}\\ \frac{\partial f}{\partial c}\end{array}\right)$$

Ecuación 4: Gradiente

$$H\,F=\left(\begin{array}{ccc}\frac{\partial^2 f}{\partial a^2} & \frac{\partial^2 f}{\partial a\,\partial b} & \frac{\partial^2 f}{\partial a\,\partial c}\\ \frac{\partial^2 f}{\partial b\,\partial a} & \frac{\partial^2 f}{\partial b^2} & \frac{\partial^2 f}{\partial b\,\partial c}\\ \frac{\partial^2 f}{\partial c\,\partial a} & \frac{\partial^2 f}{\partial c\,\partial b} & \frac{\partial^2 f}{\partial c^2}\end{array}\right)$$

Ecuación 5: Hessiano

Observemos que el gradiente se considera como vector columna y que el hessiano es una matriz simétrica. De forma abreviada, el sistema a resolver es $\nabla F=0$

4.4. Métodos numéricos: Método de Newton Raphson

4.4.1 Una variable

Supongamos que tenemos una ecuación con una incógnita $f(x)=0$ y queremos averiguar el valor de x. Si no se puede despejar la incógnita hay que recurrir a algún método numérico.

Se parte de una primera estimación de la solución x_n. Para conseguir una mejor estimación de la solución buscada se traza la tangente al punto $(x_n, f(x_n))$ y se determina su intersección con el eje x. Geométricamente, la tangente tiene pendiente $\frac{f(x_n)}{x_n - x_{n+1}}$ y es igual a la derivada $f'(x_n)$, por lo tanto

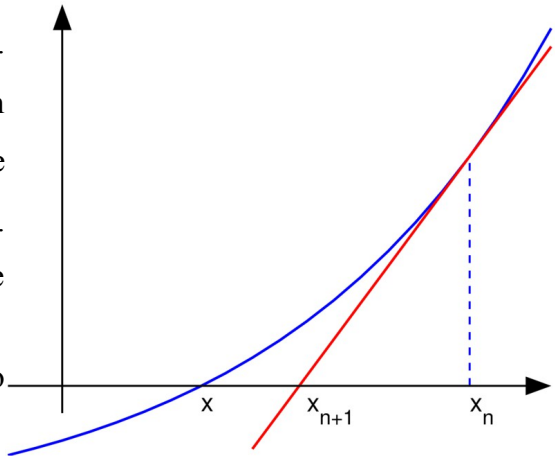


Ilustración 4: Método de Newton-Raphson

$$\frac{f(x_n)}{x_n - x_{n+1}} = f'(x_n) \rightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Ecuación 6: Newton-Raphson

Para hallar una aproximación a la solución se procede de la manera siguiente:

1. Se toma una aproximación inicial a la solución buscada x_0.
2. Utilizando la fórmula $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ se busca una aproximación mejor de la solución.
3. El proceso acaba cuando apenas hay variación entre x_n y x_{n+1}, es decir, cuando el valor $|x_n - x_{n+1}|$ es menor que una cantidad prefijada.

4.4.2 Varias variables

En el caso que nos ocupa hay que resolver el sistema $\nabla F = 0$. Por semejanza al caso de una única variable, se puede describir el método de Newton Raphson de la forma siguiente:

1. Se toma una aproximación inicial a la solución buscada $\vec{x_0}$.
2. Utilizando la fórmula $\overrightarrow{x_{n+1}} = \vec{x_n} - (H\,F(\vec{x_n}))^{-1} \cdot \nabla F(\vec{x_n})$ se busca una aproximación mejor

de la solución.

3. El proceso acaba cuando apenas hay variación entre $\vec{x_n}$ y $\overrightarrow{x_{n+1}}$, es decir, cuando el valor $|\vec{x_n} - \overrightarrow{x_{n+1}}|$ es menor que una cantidad prefijada.

Todos los vectores que aparecen se consideran vectores columna. Además, las barras del paso tres en este caso denotan la norma del vector en lugar del valor absoluto. Por último, para obtener el vector $(H\,F(\vec{x_n}))^{-1} \cdot \nabla F(\vec{x_n})$ que se puede denotar mediante $\vec{y_n}$, en lugar de invertir la matriz hessiana, es mejor resolver el sistema $(H\,F(\vec{x_n}))\vec{y_n} = \nabla F(\vec{x_n})$

5. Desarrollo

5.1. Objetos manejados

5.1.1 El plano

La ecuación del plano es $Ax+By+Cz=D$ donde A, B, C, D son coeficientes por determinar. En primer lugar, puede suponerse que el vector normal al plano es unitario ya que se puede dividir la ecuación entre $\sqrt{A^2+B^2+C^2}$. Por lo tanto, haciendo

$$a=\frac{A}{\sqrt{A^2+B^2+C^2}} \qquad b=\frac{B}{\sqrt{A^2+B^2+C^2}} \qquad c=\frac{C}{\sqrt{A^2+B^2+C^2}} \qquad d=\frac{D}{\sqrt{A^2+B^2+C^2}}$$

Ecuación 7: Coeficientes del plano

se tiene el plano $\pi: ax+by+cz=d$ con vector normal unitario, y cambiando la ecuación de signo si fuese necesario, se puede suponer que $c>0$.

Supongamos que obtenemos como resultado de la medición una serie de puntos $P_1, ..., P_n$ que queremos ajustar al plano. El mínimo error en la medición de uno de ellos provocaría que el sistema fuese incompatible, es por eso que se aplica el método de mínimos cuadrados. Para ello se calcula la distancia de cada punto $P_i=(x_i, y_i, z_i)$ al plano. Como el vector normal al plano es unitario, la distancia viene dada por $d(P_k, \pi)=\left|(a x_k+b y_k+c z_k-d)\right|$. Por simplicidad en la aplicación del método, para eliminar el valor absoluto que aparece, se toma la distancia al cuadrado. El objetivo es encontrar el valor de los parámetros a, b, c, d que hacen que la suma de los cuadrados de las distancias sea lo menor posible. Se trata por tanto de encontrar el mínimo de la función

$$F(a, b, c, d)=\sum_{k=1}^{n}(d(P_k, \pi))^2$$

Ecuación 8: Función de mínimos cuadrados

Sustituyendo el valor de la distancia queda

$$F(a, b, c, r)=\sum_{k=1}^{n}(a x_k+b y_k+c z_k-d)^2$$

Ecuación 9: Función aplicada al plano

donde ya no es necesario escribir el valor absoluto por encontrarse dicho valor elevado al cuadrado.

Para encontrar el mínimo de la función F, se deriva y se hallan sus puntos críticos. Así

$$\frac{\partial F}{\partial a}=\sum_{k=1}^{n} 2x_k(ax_k+by_k+cz_k-d)$$
$$\frac{\partial F}{\partial b}=\sum_{k=1}^{n} 2y_k(ax_k+by_k+cz_k-d)$$
$$\frac{\partial F}{\partial c}=\sum_{k=1}^{n} 2z_k(ax_k+by_k+cz_k-d)$$
$$\frac{\partial F}{\partial d}=-\sum_{k=1}^{n} 2(ax_k+by_k+cz_k-d)$$

Ecuación 10: Sistema de mínimos cuadrados del plano

Igualando a cero, obtenemos que

$$a\sum_{k=1}^{n} x_k^2+b\sum_{k=1}^{n} x_k y_k+c\sum_{k=1}^{n} x_k z_k=nd\sum_{k=1}^{n} x_k$$
$$a\sum_{k=1}^{n} x_k y_k+b\sum_{k=1}^{n} y_k^2+c\sum_{k=1}^{n} y_k z_k=nd\sum_{k=1}^{n} y_k$$
$$a\sum_{k=1}^{n} x_k z_k+b\sum_{k=1}^{n} y_k z_k+c\sum_{k=1}^{n} z_k^2=nd\sum_{k=1}^{n} z_k$$
$$a\sum_{k=1}^{n} x_k+b\sum_{k=1}^{n} y_k+c\sum_{k=1}^{n} z_k=nd$$

Ecuación 11: Sistema de mínimos cuadrados simplificado

que es un sistema lineal en las incógnitas a, b, c, d. Observemos que si $\bar{a}, \bar{b}, \bar{c}, \bar{d}$ es una solución particular y $t\in R$, entonces $t\bar{a}, t\bar{b}, t\bar{c}, t\bar{d}$ también es solución, por lo que el rango del sistema es como mucho tres. Si hacemos $nd=1$ y se toman las tres primeras ecuaciones, queda un sistema lineal

$$\begin{pmatrix} \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k y_k & \sum_{k=1}^{n} x_k z_k \\ \sum_{k=1}^{n} x_k y_k & \sum_{k=1}^{n} y_k^2 & \sum_{k=1}^{n} y_k z_k \\ \sum_{k=1}^{n} x_k z_k & \sum_{k=1}^{n} y_k z_k & \sum_{k=1}^{n} z_k^2 \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix}=\begin{pmatrix} \sum_{k=1}^{n} x_k \\ \sum_{k=1}^{n} y_k \\ \sum_{k=1}^{n} z_k \end{pmatrix}$$

Ecuación 12: Ecuación matricial del sistema

después de resolver este sistema, obtenemos el valor del vector normal, pero podría ser que su norma no fuese uno, por lo tanto, después de normalizar, se obtiene el vector normal (a,b,c). Basta hora hallar el valor de d en la última ecuación del sistema original.

El razonamiento anterior es válido cuando el plano no pasa por el origen de coordenadas, es decir, cuando d es distinto de cero.

Cuando el término independiente d es cero, hay que variar ligeramente el procedimiento. Si $d=0$ significa que el plano tiene por ecuación $ax+by+cz=0$, es decir, pasa por el origen de coordenadas. Para evitar errores al resolver el sistema, se trasladan todos los puntos una unidad en la dirección del eje x y se repiten los cálculos. De este modo se obtiene el vector normal al plano, pues este vector no cambia al hacer una traslación paralela. Si aún así el plano obtenido pasa por el origen, se repetiría el procedimiento en el eje y y en el eje z hasta obtener un plano que no pase por el origen. Una vez calculado el vector normal, el cálculo del término independiente y del punto del plano no presenta ningún problema.

5.1.2. La esfera

5.1.2.1. Ecuación exacta con cuatro puntos

La esfera tiene por ecuación $(x-a)^2+(y-b)^2+(z-c)^2=r^2$ donde (a,b,c) es el centro de la esfera y r es el radio. Como la esfera tiene cuatro parámetros (tres para el centro y el radio) hacen falta 4 puntos para determinar una esfera. Supongamos que los puntos son (x_i,y_i,z_i) con $i=1,...,4$. Se tiene así un sistema de cuatro ecuaciones con cuatro incógnitas

$$\left\{\begin{array}{l}(x_1-a)^2+(y_1-b)^2+(z_1-c)^2=r^2\\(x_2-a)^2+(y_2-b)^2+(z_2-c)^2=r^2\\(x_3-a)^2+(y_3-b)^2+(z_3-c)^2=r^2\\(x_4-a)^2+(y_4-b)^2+(z_4-c)^2=r^2\end{array}\right\}$$

Ecuación 13: Sistema de la esfera

al desarrollar los cuadrados

$$\left\{\begin{array}{l}x_1^2-2x_1a+a^2+y_1^2-2y_1b+b^2+z_1^2-2z_1c+c^2=r^2\\x_2^2-2x_2a+a^2+y_2^2-2y_2b+b^2+z_2^2-2z_2c+c^2=r^2\\x_3^2-2x_3a+a^2+y_3^2-2y_3b+b^2+z_3^2-2z_3c+c^2=r^2\\x_4^2-2x_4a+a^2+y_4^2-2y_4b+b^2+z_4^2-2z_4c+c^2=r^2\end{array}\right\}$$

Ecuación 14: Sistema de la esfera simplificado

si se resta la primera ecuación a las demás, queda

$$\left.\begin{array}{l}(x_2^2-x_1^2)-2(x_2-x_1)a+(y_2^2-y_1^2)-2(y_2-y_1)b+(z_2^2-z_1^2)-2(z_2-z_1)c=0\\(x_3^2-x_1^2)-2(x_3-x_1)a+(y_3^2-y_1^2)-2(y_3-y_1)b+(z_3^2-z_1^2)-2(z_3-z_1)c=0\\(x_4^2-x_1^2)-2(x_4-x_1)a+(y_4^2-y_1^2)-2(y_4-y_1)b+(z_4^2-z_1^2)-2(z_4-z_1)c=0\end{array}\right\}$$

Ecuación 15: Sistema obtenido restando la primera ecuación

y reorganizando los términos queda un sistema lineal de tres ecuaciones con tres incógnitas *a,b,c*

$$\left.\begin{array}{l}2(x_2-x_1)a+2(y_2-y_1)b+2(z_2-z_1)c=(x_2^2-x_1^2)+(y_2^2-y_1^2)+(z_2^2-z_1^2)\\2(x_3-x_1)a+2(y_3-y_1)b+2(z_3-z_1)c=(x_3^2-x_1^2)+(y_3^2-y_1^2)+(z_3^2-z_1^2)\\2(x_4-x_1)a+2(y_4-y_1)b+2(z_4-z_1)c=(x_4^2-x_1^2)+(y_4^2-y_1^2)+(z_4^2-z_1^2)\end{array}\right\}$$

Ecuación 16: Sistema de la esfera

que se puede resolver fácilmente mediante la regla de Cramer o el método de Gauss. Una vez que se tiene el centro (a,b,c), se sustituye en la primera ecuación y obtenemos el valor del radio *r*.

5.1.2.2. Mínimos cuadrados para la esfera

Supongamos que tenemos la esfera S de ecuación $(x-a)^2+(y-b)^2+(z-c)^2=r^2$ y hemos obtenido de forma experimental una lista de puntos. Si la medición es completamente exacta, la esfera determinada por los cuatro primeros puntos contendrá al resto de los puntos, pero el más mínimo error de medición, o simplemente la imposibilidad de tener una precisión infinita hace que al añadir más puntos el sistema sea incompatible.

Con el método de mínimos cuadrados se busca el centro (*a,b,c*) y el radio r de una esfera que mejor ajuste a la nube de puntos que se da. Para ello se calcula la distancia de cada punto $P_i=(x_i, y_i, z_i)$ a la esfera *S*. Está distancia dependerá del centro y del radio de la esfera. Por simplicidad en la aplicación del método, para eliminar las raíces cuadradas que aparecen, se toma la distancia al cuadrado. Si tenemos *n* puntos $P_1, \ldots, P_n$ el objetivo es encontrar el valor de los parámetros a, b, c, r que hacen que la suma de los cuadrados de las distancias sea lo menor posible. Se trata por tanto de encontrar el mínimo de la función

$$F(a,b,c,r)=\sum_{k=1}^{n}(d(P_k,S))^2$$

Ecuación 17: Función de mínimos cuadrados

La distancia del punto P_k a la esfera S se obtiene restando el radio a la distancia entre el punto

y el centro $d(P_k, S)=\sqrt{(x_k-a)^2+(y_k-b)^2+(z_k-c)^2}-r$ y por lo tanto la función a minimizar queda

$$F(a,b,c,r)=\sum_{k=1}^{n}(\sqrt{(x_k-a)^2+(y_k-b)^2+(z_k-c)^2}-r)^2$$

Ecuación 18: Función de mínimos cuadrados para la esfera

Estamos buscando el valor de a,b,c,r de modo que F alcance un mínimo. Por lo tanto hay que derivar la función F y encontrar sus puntos críticos. Aquí utilizamos el método de Newton-Raphson, para el cual necesitamos el gradiente ∇F y el hessiano HF. También hace falta una estimación inicial de los parámetros a,b,c,r que se obtiene a partir de los cuatro primeros puntos aplicando el método descrito en la sección anterior.

5.1.3 El cilindro

5.1.3.1. Ecuación exacta con tres puntos

El caso del cilindro es parecido al de la esfera, pero es incluso más sencillo. El cilindro tiene por ecuación $(x-a)^2+(y-b)^2=r^2$ donde (a,b) es el centro de la circunferencia que está en su base y r es el radio. Por lo tanto se tienen tres parámetros y así hacen falta 3 puntos para determinarlos. Supongamos que los puntos son (x_i, y_i, z_i) con $i=1,2,3$. Se tiene así un sistema de tres ecuaciones con tres incógnitas

$$\begin{Bmatrix}(x_1-a)^2+(y_1-b)^2=r^2\\(x_2-a)^2+(y_2-b)^2=r^2\\(x_3-a)^2+(y_3-b)^2=r^2\end{Bmatrix}$$

Ecuación 19: Sistema del cilindro

al desarrollar los cuadrados

$$\begin{Bmatrix}x_1^2-2x_1a+a^2+y_1^2-2y_1b+b^2=r^2\\x_2^2-2x_2a+a^2+y_2^2-2y_2b+b^2=r^2\\x_3^2-2x_3a+a^2+y_3^2-2y_3b+b^2=r^2\end{Bmatrix}$$

Ecuación 20: Sistema desarrollando cuadrados

si se resta la primera ecuación a las demás, queda

$$\begin{cases}(x_2^2-x_1^2)-2(x_2-x_1)a+(y_2^2-y_1^2)-2(y_2-y_1)b=0\\(x_3^2-x_1^2)-2(x_3-x_1)a+(y_3^2-y_1^2)-2(y_3-y_1)b=0\end{cases}$$

Ecuación 21: Sistema restando la primera ecuación

y reorganizando queda un sistema lineal de dos ecuaciones con dos incógnitas *a,b*

$$\begin{cases}2(x_2-x_1)a+2(y_2-y_1)b=(x_2^2-x_1^2)+(y_2^2-y_1^2)\\2(x_3-x_1)a+2(y_3-y_1)b=(x_3^2-x_1^2)+(y_3^2-y_1^2)\end{cases}$$

Ecuación 22: Sistema simplificado

que se puede resolver fácilmente mediante la regla de Cramer o el método de Gauss. Una vez que se tiene el centro (*a,b*), se sustituye en la primera ecuación y obtenemos el valor del radio *r*.

5.1.3.2. Mínimos cuadrados para el cilindro

Como para la esfera, si tenemos una nube de *n* puntos $P_1,...,P_n$ el objetivo es encontrar el valor de los parámetros a,b,r que hacen que la suma de los cuadrados de las distancias sea lo menor posible. Se trata por tanto de encontrar el mínimo de la función

$$F(a,b,c,r)=\sum_{k=1}^{n}(d(P_k,C_{a,b,r}))^2$$

Ecuación 23: Función de mínimos cuadrados

donde $C_{a,b,r}:(x-a)^2+(y-b)^2=r^2$ es el cilindro cuya circunferencia base tiene centro (*a,b*) y radio *r*. La distancia del punto P_k al cilindro C se obtiene con la fórmula $d(P_k,C)=\sqrt{(x_k-a)^2+(y_k-b)^2}-r$ y por lo tanto la función a minimizar queda

$$F(a,b,r)=\sum_{k=1}^{n}(\sqrt{(x_k-a)^2+(y_k-b)^2}-r)^2$$

Ecuación 24: Función de mínimos cuadrados simplificada

De nuevo hay que derivar la función F y encontrar sus puntos críticos. Aquí se utiliza el método de Newton-Raphson, para el cual se necesita el gradiente ∇F y el hessiano HF. También hace falta una estimación inicial de los parámetros a,b,r que se obtiene a partir de los tres primeros puntos aplicando el método descrito en la sección anterior.

5.1.4 El cono

Para el cono se utiliza la ecuación $(x-a)^2+(y-b)^2=d(z-c)^2$ donde (a,b,c) es el vértice del cono y el parámetro d determina el ángulo de apertura. Se trata de un cono vertical, es decir, su eje es paralelo al eje z.

Supongamos que tenemos *n* puntos $P_1, \ldots, P_n$ el objetivo es encontrar el valor de los parámetros a, b, c, d que determinan un cono que mejor se ajusta a la nube de puntos dada. En este caso no se utiliza el método de mínimos cuadrados. En primer lugar se necesita localizar el eje del cono y para ello se manipulan las ecuaciones hasta conseguir un sistema lineal.

En principio el sistema tiene la forma

$$\begin{Bmatrix} (x_1-a)^2+(y_1-b)^2=d\,(z_1-c)^2 \\ (x_2-a)^2+(y_2-b)^2=d\,(z_2-c)^2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ (x_n-a)^2+(y_n-b)^2=d\,(z_n-c)^2 \end{Bmatrix}$$

Ecuación 25: Sisteam del cono

si se desarrollan los cuadrados queda

$$\begin{Bmatrix} x_1^2-2\,x_1\,a+a^2+y_1^2-2\,y_1\,b+b^2=d\,z_1^2-2\,z_1\,d\,c+d\,c^2 \\ x_2^2-2\,x_2\,a+a^2+y_2^2-2\,y_2\,b+b^2=d\,z_2^2-2\,z_2\,d\,c+d\,c^2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n^2-2\,x_n\,a+a^2+y_n^2-2\,y_n\,b+b^2=d\,z_n^2-2\,z_n\,d\,c+d\,c^2 \end{Bmatrix}$$

Ecuación 26: Sistema desarrollando cuadrados

restando la primera ecuación a todas las demás queda

$$\begin{Bmatrix} (x_2^2-x_1^2)-2\,(x_2-x_1)\,a+(y_2^2-y_1^2)-2\,(y_2-y_1)\,b=d\,(z_2^2-z_1^2)-2\,(z_2-z_1)\,d\,c \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ (x_n^2-x_1^2)-2\,(x_n-x_1)\,a+(y_n^2-y_1^2)-2\,(y_n-y_1)\,b=d\,(z_n^2-z_1^2)-2\,(z_n-z_1)\,d\,c \end{Bmatrix}$$

Ecuación 27: Sistema restando la primera ecuación

que es un sistema lineal si hacemos el cambio $c'=dc$

$$\begin{Bmatrix} 2\,(x_2-x_1)\,a+2\,(y_2-y_1)\,b-2\,(z_2-z_1)\,c'+(z_2^2-z_1^2)\,d=(x_2^2-x_1^2)+(y_2^2-y_1^2) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ 2\,(x_n-x_1)\,a+2\,(y_n-y_1)\,b+2\,(z_n-z_1)\,c'+(z_n^2-z_1^2)\,d=(x_n^2-x_1^2)+(y_n^2-y_1^2) \end{Bmatrix}$$

Ecuación 28: Sistema haciendo un cambio

Este nuevo sistema tiene $n-1$ ecuaciones y 4 incógnitas y puede resolverse por el método de

mínimos cuadrados para sistemas lineales. Una vez resuelto, se deshace el cambio para obtener el valor de c .

Este método de resolución toma el primer punto como punto base para el cálculo ya que la primera ecuación se resta a todas las demás. Para no permitir que el primer punto tenga una posición privilegiada y que pequeños errores en la medición del primer punto tengan una repercusión mayor que en el resto de los puntos, el programa realiza el cálculo anterior n veces, cogiendo cada vez un punto distinto como primer punto. Después realiza la media aritmética de los valores de los parámetros. Recordemos que en otros programas similares también se considera la estrategia de calcular el promedio de los parámetros como una estrategia válida.

Sin embargo, a pesar de que el método anterior da los cuatro parámetros del cono necesarios, se observa que ligeras variaciones en estos parámetros pueden producir cambios significativos en la altura del vértice, es decir, en el parámetro *c*. Es por ello que para obtener un mejor ajuste se ha variado el procedimiento de la siguiente manera.

Las coordenadas (*a,b*) de la proyección del vértice son muy precisas, pero para obtener la coordenada *c* que mejor ajuste se ha hecho una traslación para que el eje del cono coincida con el eje *z*, es decir, se ha restado (*a,b*,0) a cada punto. Después se han tomado nuevas coordenadas sobre el primer cuadrante $\bar{x}=\sqrt{x^2+y^2}$ e $\bar{y}=z$. De esta forma se tiene una nube de puntos que se ajsuta mediante la recta de regresión. La gráfica muestra la nube de puntos que se ha obtenido en uno de los conos que se han medido con la máquina

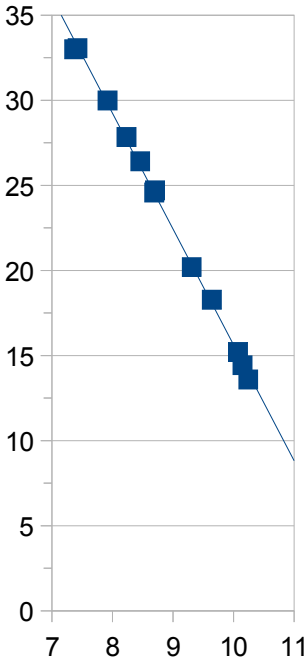


Ilustración 5: Recta de regresión

Si los puntos tienen coordenadas (x_i, y_i) para ajustarlos mediante la recta de regresión $y = a\,x + b$ hay que minimizar la función $f(a,b) = \sum (y_i - a\,x_i - b)^2$ para lo cual se toman sus derivadas parciales y se iguala a cero

$$\left.\begin{array}{l} \dfrac{\partial f}{\partial a} = \sum -2\,x_i (y_i - a\,x_i - b) = 0 \\ \dfrac{\partial f}{\partial b} = \sum -2 (y_i - a\,x_i - b) = 0 \end{array}\right\} \rightarrow \begin{bmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & n \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum x_i \cdot y_i \\ \sum y_i \end{bmatrix}$$

Ecuación 29: Sistema para la recta de regresión

de donde

$$a = \frac{n \cdot \sum x_i \cdot y_i - \sum x_i \cdot \sum y_i}{n \cdot \sum x_i^2 - \sum x_i \cdot \sum x_i} \qquad b = \frac{\sum x_i^2 \cdot \sum y_i - \sum x_i \cdot \sum x_i \cdot y_i}{n \cdot \sum x_i^2 - \sum x_i \cdot \sum x_i}$$

Ecuación 30: Soluciones del sistema

Ahora la coordenada *z* del vértice se obtiene haciendo $x = 0$ en $y = a\,x + b$ y resulta ser b y el radio del cono en el plano xy se obtiene haciendo $y = 0$ en la misma recta por lo que es $-b/a$.

Para obtener los parámetros buscados, es decir, para ajustar la nube de puntos a la ecuación $(x-a)^2 + (y-b)^2 = d\,(z-c)^2$ solo queda explicar como se obtiene d a partir del radio r y de

la altura del vértice c . Haciendo $z=0$ en la ecuación obtenemos $d\cdot c^2=r^2\rightarrow d=\frac{r^2}{c^2}$.

Para calcular el máximo error cometido, se hace sobre el plano $\bar{x}\,\bar{y}$, para ello basta calcular el máximo de los $|y_i-ax_i-b|$.

5.1.5. El cilindro girado

Para poder calcular los parámetros del cilindro, se necesita un plano auxiliar que se supondrá que es perpendicular al eje del cilindro.

Dado el plano auxiliar, se calcula el vector normal $\vec{u}'=(u'_1,u'_2,u'_3)$ y se hace un giro para transformar este vector en el vector $\vec{k}=(0,0,1)$. Para ello se necesita un vector perpendicular a $\vec{u}'$ y basta coger el vector $\vec{v}'=(-u'_2,u'_1,0)$. Hace falta un tercer vector perpendicular a ambos y para ello se toma el producto vectorial $\vec{w}=\vec{u}\times\vec{v}$ que si desarrollamos se tiene

$$\vec{w}=\begin{vmatrix}\vec{i} & \vec{j} & \vec{k}\\ u'_1 & u'_2 & u'_3\\ -u'_2 & u'_1 & 0\end{vmatrix}=(-u'_1\cdot u'_3)\vec{i}+(-u'_2\cdot u'_3)\vec{j}+(u'^2_1+u'^2_2)\vec{k}=(-u'_1\cdot u'_3,-u'_2\cdot u'_3,u'^2_1+u'^2_2)$$

Ecuación 31: Producto vectorial

Para obtener una base ortonormal, se divide cada uno de los vectores anteriores entre su norma y tenemos así

$$\vec{u}=\frac{\vec{u}'}{\|\vec{u}'\|}\qquad \vec{v}=\frac{\vec{v}'}{\|\vec{v}'\|}\qquad \vec{w}=\frac{\vec{w}'}{\|\vec{w}'\|}$$

Ecuación 32: Base ortonormal

Si las coordenadas de estos vectores son $\vec{u}=(u_1,u_2,u_3)$, $\vec{v}=(v_1,v_2,v_3)$ (aquí $v_3=0$ pero no es necesario) , $\vec{w}=(w_1,w_2,w_3)$ entonces la matriz que tiene estos vectores por columnas

$$A=\begin{pmatrix} u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \\ u_3 & v_3 & w_3 \end{pmatrix}$$

Ecuación 33:
Matriz de giro

realiza un giro que transforma los vectores de la base canónica $\{\vec{i}, \vec{j}, \vec{k}\}$ en los vectores de la base $\{\vec{u}, \vec{v}, \vec{w}\}$. Su matriz inversa por lo tanto transforma la nube de puntos del cilindro girado en una nube de puntos correspondiente a un cilindro que reposa sobre el plano $z=0$, a la que se le puede aplicar el procedimiento anterior. Después basta con multiplicar por la matriz A el centro de la circunferencia sobre el plano $z=0$ que proyecta este cilindro para obtener un punto del eje del cilindro girado. Los radios de ambos cilindros coinciden y la dirección del eje del cilindro girado es el vector $\vec{u}$.

En resumen, se calcula la matriz A y su matriz inversa A^{-1} que coincide con su traspuesta puesto que ambas son ortonormales. A continuación se multiplica cada uno de los puntos de la nube por la matriz A^{-1} para obtener una nueva nube de puntos. Se calculan luego los parámetros correspondientes a esta nueva nube de puntos y se deshace el cambio de coordenadas con ayuda de la matriz A.

El proceso anterior falla si $u'_1 = u'_2 = 0$ porque en ese caso $\vec{v}'$ sería igual a cero. Sin embargo, basta tomar cualquier otro vector perpendicular a $\vec{u}'$ y repetir el proceso anterior. Por ejemplo, bastaría con tomar $\vec{v}' = (0, -u'_3, u'_2)$ en cuyo caso $\vec{w}' = (u'^2_2 + u'^2_3, -u'_1 \cdot u'_2, -u'_1 \cdot u'_3)$

5.1.6. El cono girado

La situación es análoga a la del cilindro girado. Se transforma la nube de puntos, se calculan los parámetros y se deshace el cambio.

5.2. Ajuste del radio del palpador

El palpador manejado por el programa tiene un radio de 1mm que queda almacenado en la constante RPALPADOR. Esta distancia se utiliza en cada uno de los objetos calculados para realizar

el ajuste correspondiente. Hay dos formas de realizar el ajuste: indicando si el palpador ha realizado la medición por el exterior o el interior del objeto o indicando un punto adicional que denominaremos “disparo al aire”. También se ha añadido una opción para calcular los parámetros del objeto ideal, sin tener en cuenta ningún ajuste, ya que resultaba de utilidad para comprobar los resultados obtenidos.

5.2.1. Plano

En el caso de la esfera, el cilindro o el cono, resulta evidente que se entiende por parte externa o interna del objeto. Sin embargo, dado un plano, no resulta tan claro a cual de los dos semiespacios resultantes llamar parte externa y parte interna. Es por ello que en el caso del plano, si se marca la posición del palpador como externa o interna el efecto del programa será el mismo: despreciará el radio del palpador y dará los parámetros del plano ideal calculado. El plano original es un plano paralelo a él y separado por 1 mm de distancia, pero no queda clara cuál de las dos opciones posibles es.

Para especificar completamente el plano palpado sólo se considerará la opción del “disparo al aire”. Al hacer un disparo al aire se está fijando el semiespacio sobre el que se ha movido el palpador para hacer la medición. Si un punto del plano es $P=(x, y, z)$ y el punto que da el disparo al aire es $Q=(\bar{x}, \bar{y}, \bar{z})$ y el vector normal es $\vec{n}$, basta considerar el signo del producto escalar $\overrightarrow{PQ}\cdot\vec{n}$ para saber en que dirección hay que mover el plano ideal para obtener el plano real.

- Si $\overrightarrow{PQ}\cdot\vec{n}>0$ el un punto del plano real es $P-\vec{n}$.
- Si $\overrightarrow{PQ}\cdot\vec{n}<0$ el un punto del plano real es $P+\vec{n}$.

El vector normal en cualquiera de los dos casos es el mismo, $\vec{n}$.

5.2.2. Esfera

Si la esfera se palpa desde el exterior, basta restar RPALPADOR al radio para obtener el parámetro real. El centro sigue siendo el mismo. Si por el contrario se palpa desde el interior, basta sumar RPALPADOR al radio.

Si se realiza un disparo al aire, hay que comprobar si la distancia entre el centro y el disparo al aire Q es mayor que el radio ideal, en cuyo caso se estaría palpando desde el exterior o bien si es menor que el radio ideal se estaría palpando desde el interior.

5.2.3. Cilindro

Este caso es prácticamente igual al de la esfera.

5.2.4. El cono

En realidad se trata de una superficie que es medio cono, y no se admite que se tomen puntos del cono completo y tampoco se admite que uno de los puntos introducidos sea el vértice del cono. Ahora hay que distinguir dos casos, según el medio cono apunte a la parte positiva del eje z o a la parte negativa. Para saber en cual de los dos casos estamos, bastará considerar el signo de la diferencia entre las terceras coordenadas de un punto cualquiera de los medidos y el vértice del cono ideal.

Supongamos primero que el palpado es externo. En ese caso, de acuerdo con la geometría de la figura

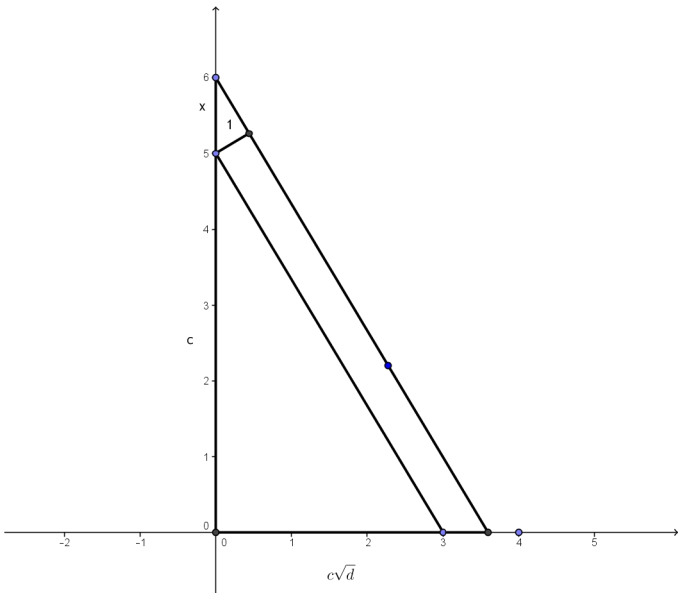


Ilustración 6: Ajuste del palpador en el cono

se tiene que por semejanza de triángulos $\frac{x}{1}=\frac{\sqrt{\left(c\sqrt{d}\right)^2+c^2}}{c\sqrt{d}}$ de donde $x=\sqrt{\frac{d+1}{d}}$ y es ésta la cantidad que hay que añadir a la tercera coordenada si el palpado es externo (se ha supuesto que el palpador tiene radio uno, en otro caso basta multiplicar esa cantidad por el radio del palpador).

Cuando el palpado es interno, se procede de forma análoga, sólo que esta vez hay que restar dicha cantidad de la tercera coordenada.

5.3. Cálculo de distancias, puntos, rectas y ángulos

5.3.1. Distancias

Hay cuatro fórmulas básicas que se van a utilizar que son las que dan la distancia entre dos puntos, entre un punto y una recta, entre un punto y un plano y entre dos rectas. Estas fórmulas son

- Punto-punto $d(P,Q)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+(z_1-z_2)^2}$ si $P=(x_{1,}y_{1,}z_1)$ y $Q=(x_{2,}y_{2,}z_2)$.
- Punto-plano $d(P,\pi)=\frac{|(ax+by+cz-d)|}{\sqrt{a^2+b^2+c^2}}$ si $P=(x_{1,}y_{1,}z_1)$ y $\pi: a\,x+b\,y+c\,z=d$
- Punto-recta $d(P,r)=\frac{|(\overrightarrow{PQ}\cdot\vec{d})|}{||\vec{d}||}$ si $Q\in r$ y d es un vector dirección de r .
- Recta-recta $d(r,s)=\frac{|[PQ,\vec{d}_{1,}\vec{d}_2]|}{||\vec{d}_1\times\vec{d}_2||}$ donde $P\in r$, $Q\in s$ $\vec{d}_1$ y $\vec{d}_2$ son vectores directores de r y s .

Ahora las distancias posibles que calcula el programa se pueden resumir en la tabla siguiente

	Esfera	Cilindro	Cono	Plano	Punto	Recta
Esfera	P-P	P-R	P-P, P-R	P-pl	P-P	P-R
Cilindro		R-R	P-R,R-R		P-R	R-R
Cono			P-P,P-R,R-R	P-pl	P-P, P-R	P-R, R-R
Plano					P-pl	
Punto				P-pl	P-P	P-R
Recta						R-R

P=punto R=recta pl=plano

Se ha evitado calcular distancias de objetos para los que solo existe la distancia cuando son paralelos, por ejemplo, plano-plano y recta-plano.

5.3.2. Intersecciones

El programa calcula las siguientes intersecciones:

1. Plano-Plano: da lugar a una recta.
2. Plano-Recta: da lugar a un punto.

5.3.2.1 Plano-plano

Si los planos tienen puntos $P_1=(x_1, y_1, z_1)$ y $P_2=(x_2, y_2, z_2)$ y vectores normales $\vec{n}_1=(u_1, v_1, w_1)$ y $\vec{n}_2=(u_2, v_2, w_2)$ entonces la recta intersección se obtiene a partir de un punto que esté en los dos planos y del vector dirección que es el producto vectorial $\vec{d}:=\vec{n}_1\wedge\vec{n}_2$. Para obtener un punto $P_3:=(x_3, y_3, z_3)$ que esté en ambos planos el programa da en primer lugar el valor cero a la coordenada x e intenta resolver el sistema formado por los tres planos. Si esto no es posible, repite el proceso con la coordenada y y con la coordenada z .

5.3.2.2 Plano-recta

Si la recta tiene punto $P=(a, b, c)$ y vector dirección $\vec{d}=(u, v, w)$ y el plano tiene por ecuación $\pi: Ax+By+Cz+D=0$ entonces el punto de intersección $Q=P+t\vec{d}$ con $t\in R$ se obtiene al resolver la ecuación $A(a+tu)+B(b+tv)+C(c+tw)+D=0$ de donde se obtiene $t=\frac{-D-Aa-Bb-Cc}{Au+Bv+Cw}$. Ahora basta sustituir en $Q=P+t\vec{d}$ para obtener dicho punto. Esto es posible siempre que la recta y el plano no sean paralelos, es decir, cuando el producto escalar $\vec{d}\cdot\vec{n}\neq 0$ ya que es precisamente este el denominador que aparece al despejar t .

5.3.3. Ángulos

El programa calcula los siguientes ángulos

- Semiángulo del cono.
- Ángulo entre dos planos

- Ángulo entre una recta y un plano.

5.3.3.1 Ángulo entre dos planos

El programa calcula el ángulo entre los vectores normales de los dos planos aplicando la fórmula del producto escalar. Si los vectores normales son $\vec{n}_1$ y $\vec{n}_2$ entonces $\cos\alpha = \frac{|\vec{n}_1 \cdot \vec{n}_2|}{\|\vec{n}_1\| \cdot \|\vec{n}_2\|}$. El cálculo de dicho ángulo se efectúa a la vez que la recta intersección, pero el resultado aparece únicamente en una ventana emergente.

5.3.3.2 Semiángulo del cono

Al calcular la intersección del cono con un plano vertical que pase por el vértice obtenemos un triángulo isósceles (ver la ilustración 6). Si trazamos la altura de este triángulo vemos que $\tan\alpha = \frac{c\sqrt{d}}{c} = \sqrt{d}$ y por lo tanto $\arctan(\sqrt{d}) = \alpha$.

5.4. Descripción de los tipos de datos manejados

La nube de puntos puede introducirse directamente en el programa a través de la matriz habilitada para tal efecto. También están permitidas otras opciones como hacer un corta y pega desde una hoja de cálculo o cargar un archivo con la nube de puntos. La estructura de un archivo de puntos queda reflejada en el siguiente archivo de ejemplo

```
x;y;z
-75,501;-95,271;-30,138
-54,968;-99,466;-31,650
-80,055;-89,773;-29,379
-81,857;-76,261;-26,801
-72,481;-61,070;-31,450
-55,628;-59,456;-30,368
-42,904;-70,348;-35,430
;;
;;
;;
;;
;;
;;
;;
;;
;;
;;
;;
;;
;;
```

Como puede verse se utiliza el punto y coma como separador entre campos, la coma como separador decimal y el fin de línea para una nueva línea.

La lista de objetos que crea el programa también se puede guardar en un archivo que sigue el mismo formato. Es un archivo de texto que utiliza el punto y coma como separador de celdas y el fin de línea para la nueva línea.

El formato es muy simple y accesible para facilitar futuras ampliaciones o la integración del programa con otros proyectos.

5.5. Funcionamiento del programa

El programa obtenido consta de dos archivos principales a los que hay que añadir los archivos que produce *Lazarus*. El primero de los archivos de código contiene 1350 líneas mientras que el segundo de los archivos contiene 2222 líneas. Una vez compilado se obtiene un único archivo ejecutable de casi 17 megas.

El funcionamiento del programa a nivel de usuario se describe en el manual de usuario que se incluye como anexo.

5.6. Validación del programa

Para comprobar la corrección con la que el programa realiza los cálculos se han hecho pruebas periódicas. En principio la lista de puntos para cada prueba se obtenía introduciendo las ecuaciones del objeto correspondiente en la hoja de cálculo en lugar de realizar una medición física con la propia máquina.

A continuación se muestra una de las pruebas realizadas para el cálculo de los parámetros del cono girado.

La lista de puntos corresponde al cono recto (eje paralelo al eje *z*) y el vértice que aparece es el vértice correspondiente al cono recto una vez que se ha realizado el ajuste correspondiente al radio del palpador. La matriz de giro se ha utilizado para obtener los puntos de la segunda tabla, que forman la lista de puntos del cono girado que se introducen en el programa. Por último aparece la lista de puntos del plano perpendicular al eje del cono girado.

	x	y	z		x	y	z
	4	2	6		-0,180822091	5,476032159	5,097094777
	1	0	5		2,1796445648	3,5938581271	2,8867513459
	3	2	5		-0,051065579	4,1915751086	4,5197445078
	1	1	4		1,1940460051	2,6082595675	3,1258976577
	4	-2	8		2,6068716097	8,2637258592	2,9858089917
	13	7	16		-2,812521882	15,572254429	14,953080374
vértice	1	2	4,41		1,022511325	2,4367248874	4,179107849

Ilustración 7: Nube de puntos

1	2	-3
5	3	-8
3	7	-10
4	2	-6
8	1	-9

Ilustración 8: Plano perpendicular al eje

La lista de puntos se copia y se pega en el programa, así como la lista de puntos del plano auxiliar.

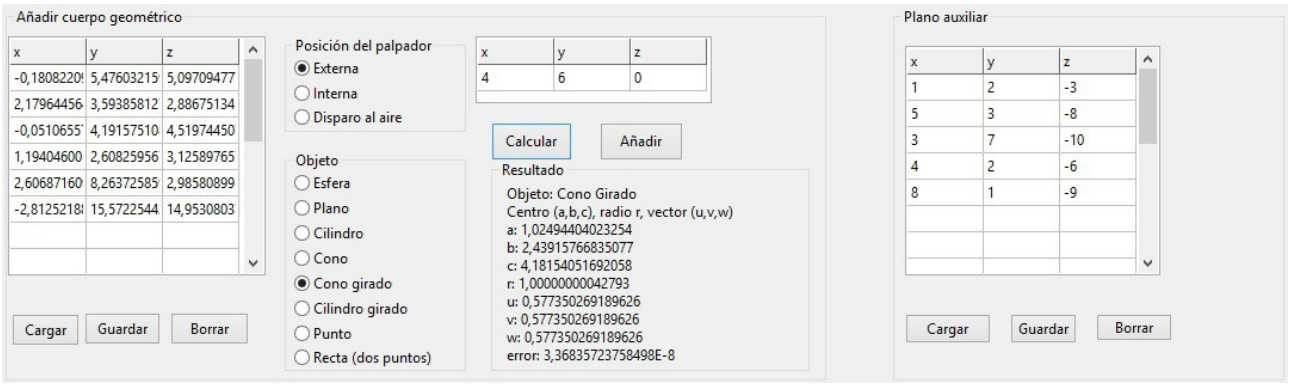


Ilustración 9: Cono girado

El resultado del cálculo realizado por el programa se corresponde a lo esperado tal y como muestra la ilustración 10.

Ilustración 10: Resultados del cono girado

5.6.1 Plano

Tomando datos reales del programa tenemos la siguiente nube de puntos

x	y	z
-24,718	-5,069	-38,863
-48,766	3,189	-24,05
-66,366	-21,28	-13,357
-57,174	-46,728	-19,166
-37,667	-48,443	-31,183
-16,761	-48,452	-44,046

Ilustración 11: Nube de puntos del plano

El programa proporciona los siguientes resultados

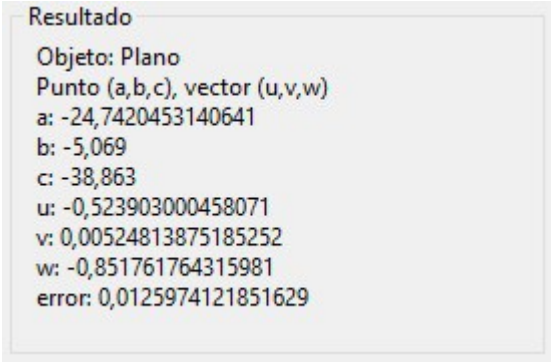


Ilustración 12: Resultado del plano

En la ilustración 13 se muestran la nube de puntos y el plano ideal calculado.

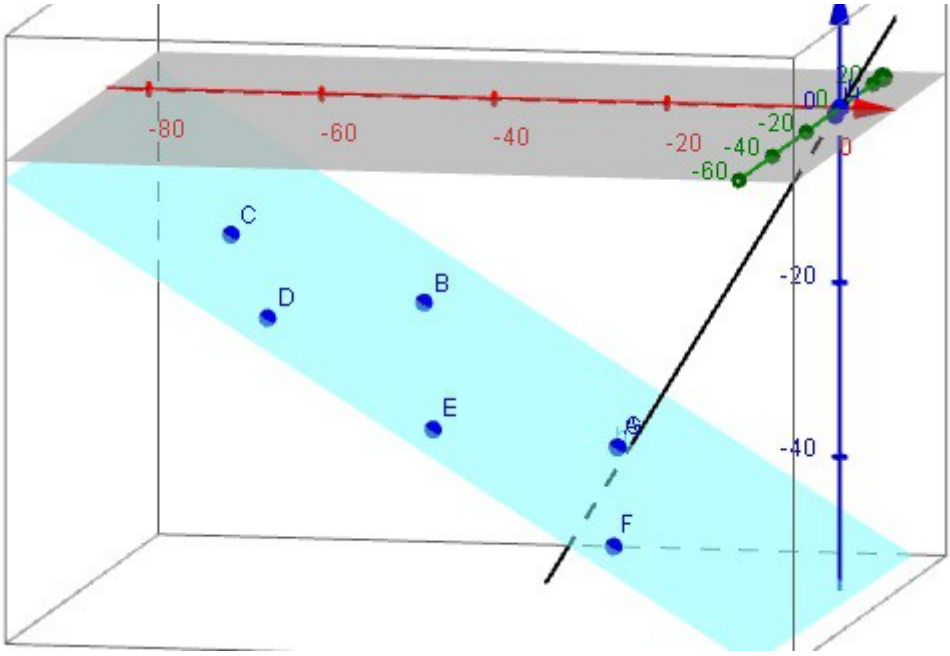


Ilustración 13: Representación del plano

5.6.2 Esfera

Los datos introducidos en el programa han sido

x	y	z
-175,354	128,52	98,715
-179,274	149,085	105,897
-180,87	128,879	106,086
-184,562	146,503	107,058
-182,445	130,129	106,146
-178,057	134,767	112,224
-170,215	132,018	97,228

Ilustración 14: Nube de puntos de la esfera

Los resultados proporcionados por el programa son

Ilustración 15: Resultado de la esfera

La nube de puntos aparece a continuación junto con la esfera ideal calculada.

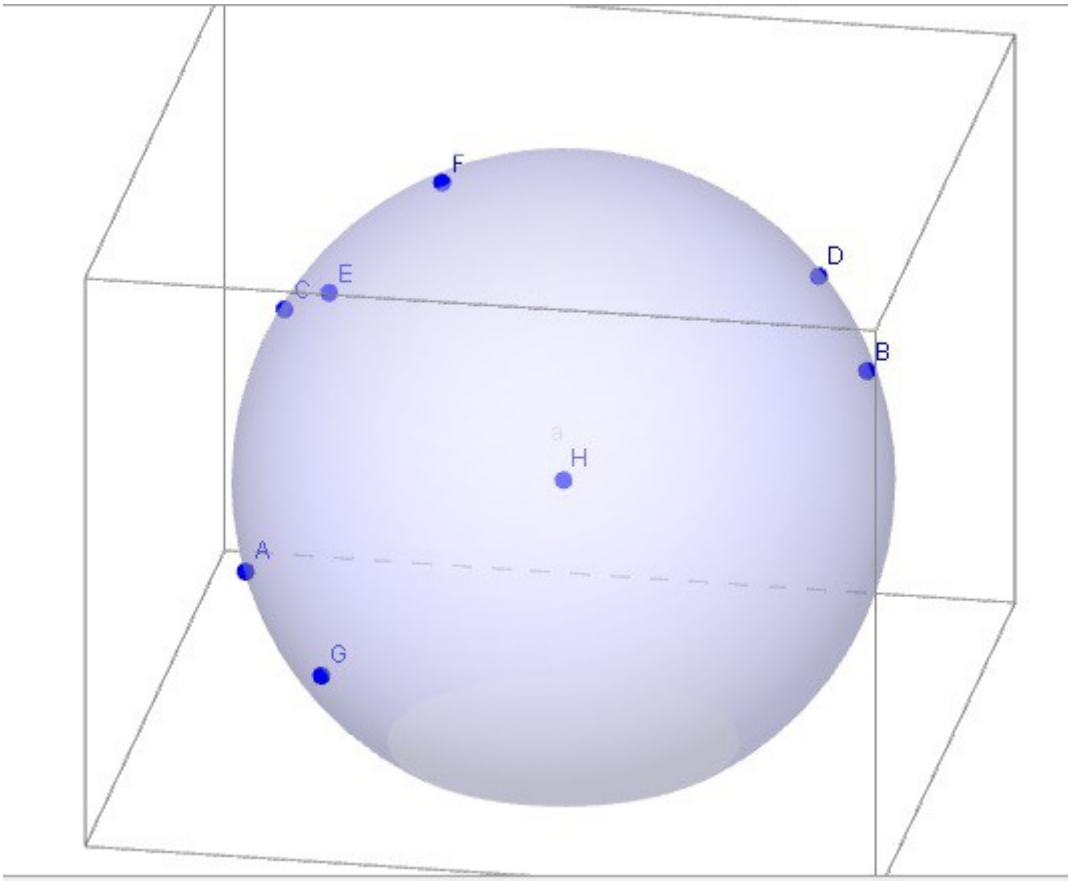


Ilustración 16: Representación de la esfera

5.6.3 Cilindro vertical 1

Tomando datos reales del programa tenemos la siguiente nube de puntos

x	y	z
15,205	58,144	-46,798
10,764	61,738	-45,749
15,943	67,199	-46,906
19,799	61,466	-46,189
12,098	65,827	-46,817
19,003	60,004	-47,291
11,174	60,931	-46,999
17,96	66,524	-46,631

Ilustración 17: Nube de puntos del cilindro

Para la cual el programa proporciona los siguientes resultados

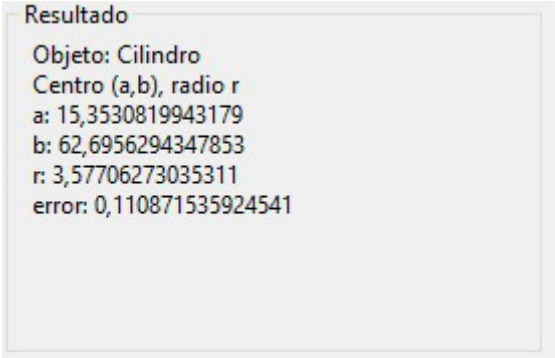


Ilustración 18: Resultado del cilindro

Hemos supuesto que el palpado es exterior al cilindro. A continuación se muestra la nube de puntos proyectada sobre el plano *xy* así como las dos circunferencias, la ideal y la real que se obtiene restando el radio del palpador

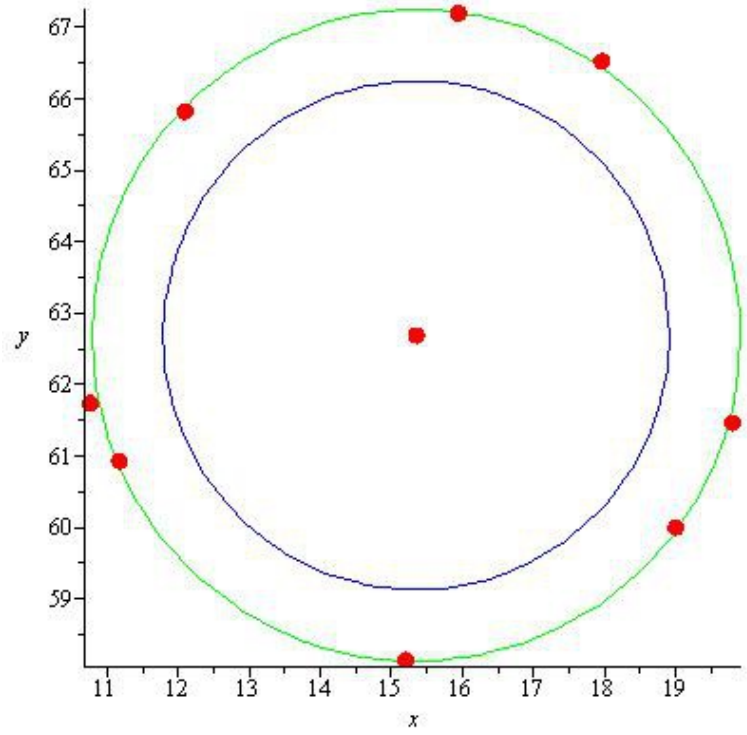


Ilustración 19: Representación del cilindro

5.6.4 Cilindro vertical 2

Tomando datos reales del programa tenemos la siguiente nube de puntos

x	y	z
28,394	78,287	-39,809
48,972	54,511	-40,744
21,515	57,585	-39,686
20,53	68,047	-39,117
51,926	68,044	-38,848
48,211	75,009	-41,03
29,41	49,623	-40,991
48,712	54,161	-40,222
27,246	77,549	-40,281

Ilustración 20: Nube de puntos del cilindro

Para la cual el programa proporciona los siguientes resultados

Ilustración 21: Resultado del cilindro

Se ha supuesto que el palpado es exterior al cilindro. A continuación se muestra la nube de puntos proyectada sobre el plano *xy* así como las dos circunferencias, la ideal y la real que se obtiene restando el radio del palpador

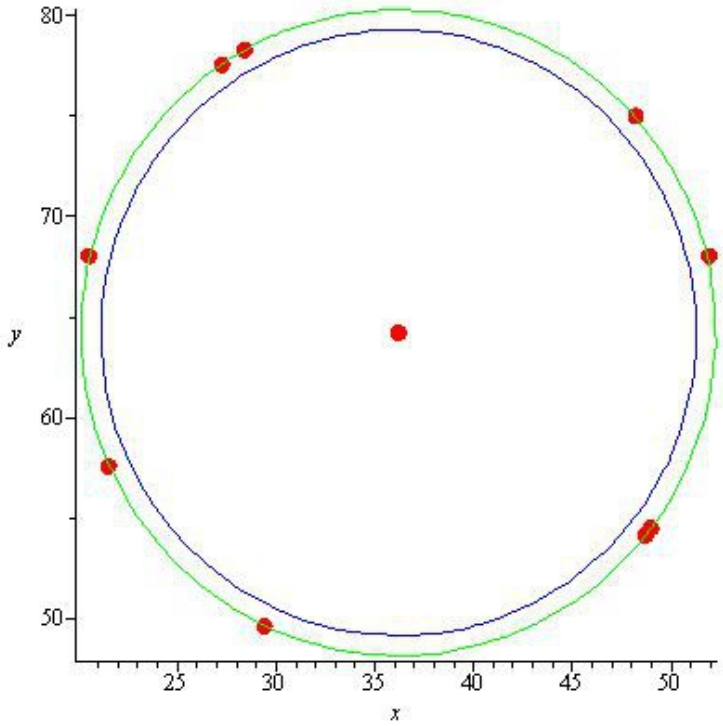


Ilustración 22: Representación del cilindro

5.6.5 Cilindro girado

Para comprobar el funcionamiento del programa aplicado a un cilindro girado, se utilizó una pieza como la que aparecen en la ilustración 23.

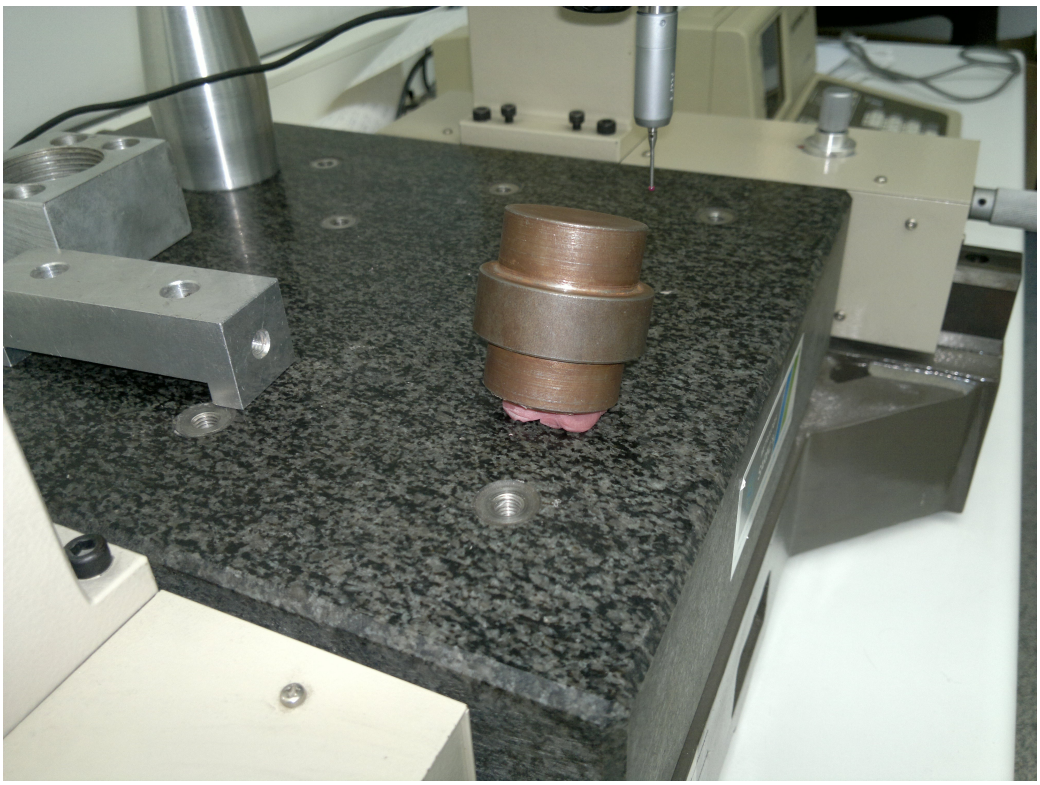


Ilustración 23: Cilindro girado de pruebas

A partir de la nube de puntos

Cilindro girado 1		
x	y	z
-75,501	-95,271	-30,138
-54,968	-99,466	-31,65
-80,055	-89,773	-29,379
-81,857	-76,261	-26,801
-72,481	-61,07	-31,45
-55,628	-59,456	-30,368
-42,904	-70,348	-35,43

Plano auxiliar		
x	y	z
-48,841	-81,016	-30,1
-50,988	-69,07	-28,89
-61,529	-69,073	-25,297
-69,523	-76,27	-22,842
-71,43	-87,848	-22,638
-62,231	-82,236	-25,59

Ilustración 24: Nube de puntos del cilindro

Se obtienen los siguientes resultados

Resultado

Objeto: Cilindro girado
Centro (a,b,c), radio r, direccion (u,v,w)
a: -46,4674978739654
b: -81,2161671156084
c: 12,6853757476626
r: 19,9363036780135
u: -0,323295575930972
v: 0,0372821131120142
w: -0,945563331895524
error: 0,144014344739826

Ilustración 25: Resultado del cilindro

A continuación se muestra la nube de puntos y el cilindro que las aproxima.

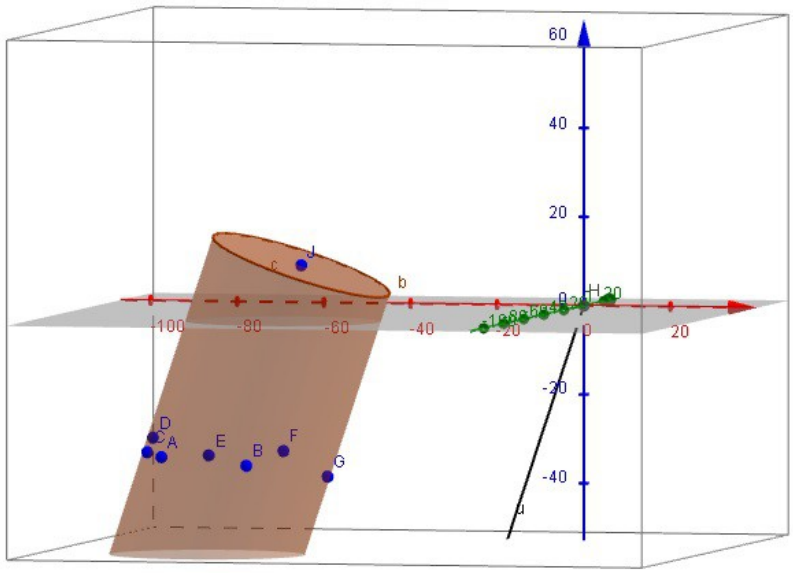


Ilustración 26: Representacion del cilindro

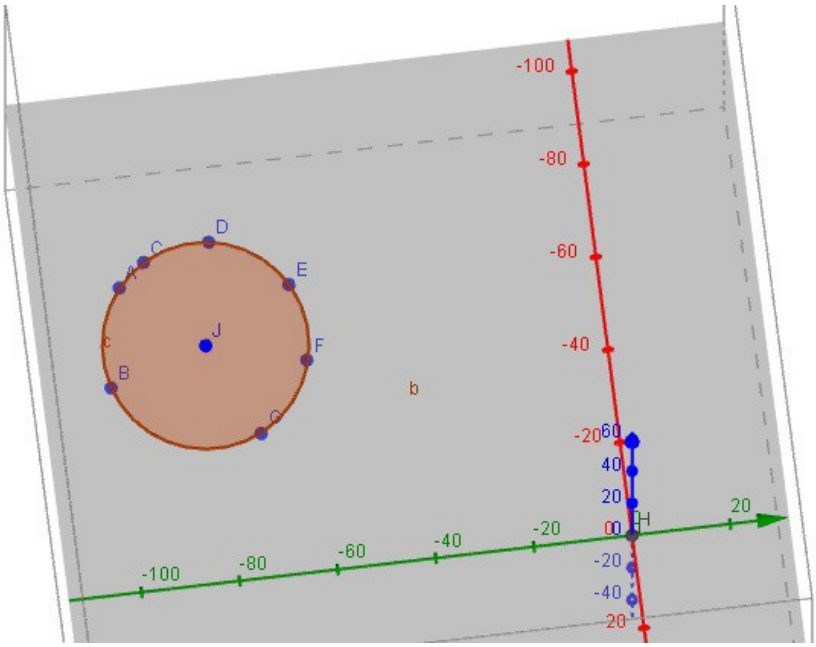


Ilustración 27: Vista superior

Si ahora se añaden los puntos de otro cilindro concéntrico

x	y	z
-60,759	-104,019	-48,666
-73,753	-103,747	-43,21
-86,732	-97,926	-51,663
-90,744	-90,391	-44,966
-94,957	-77,198	-48,859
-87,875	-67,037	-36,801
-80,604	-57,168	-42,598
-65,277	-53,101	-47,275
-49,713	-59,824	-49,84

Ilustración 28: Nube de puntos del otro cilindro

El programa proporciona los resultados siguientes

Resultado

Objeto: Cilindro girado
Centro (a,b,c), radio r, direccion (u,v,w)
a: -46,6404553702664
b: -81,2568418712447
c: 12,7429075392442
r: 24,8828553347372
u: -0,323295575930972
v: 0,0372821131120142
w: -0,945563331895524
error: 0,0769792586215985

Ilustración 29: Resultado del otro cilindro

Si se representan las dos nubes de puntos sobre los mismos ejes con los cilindros correspondientes obtenemos las siguientes figuras

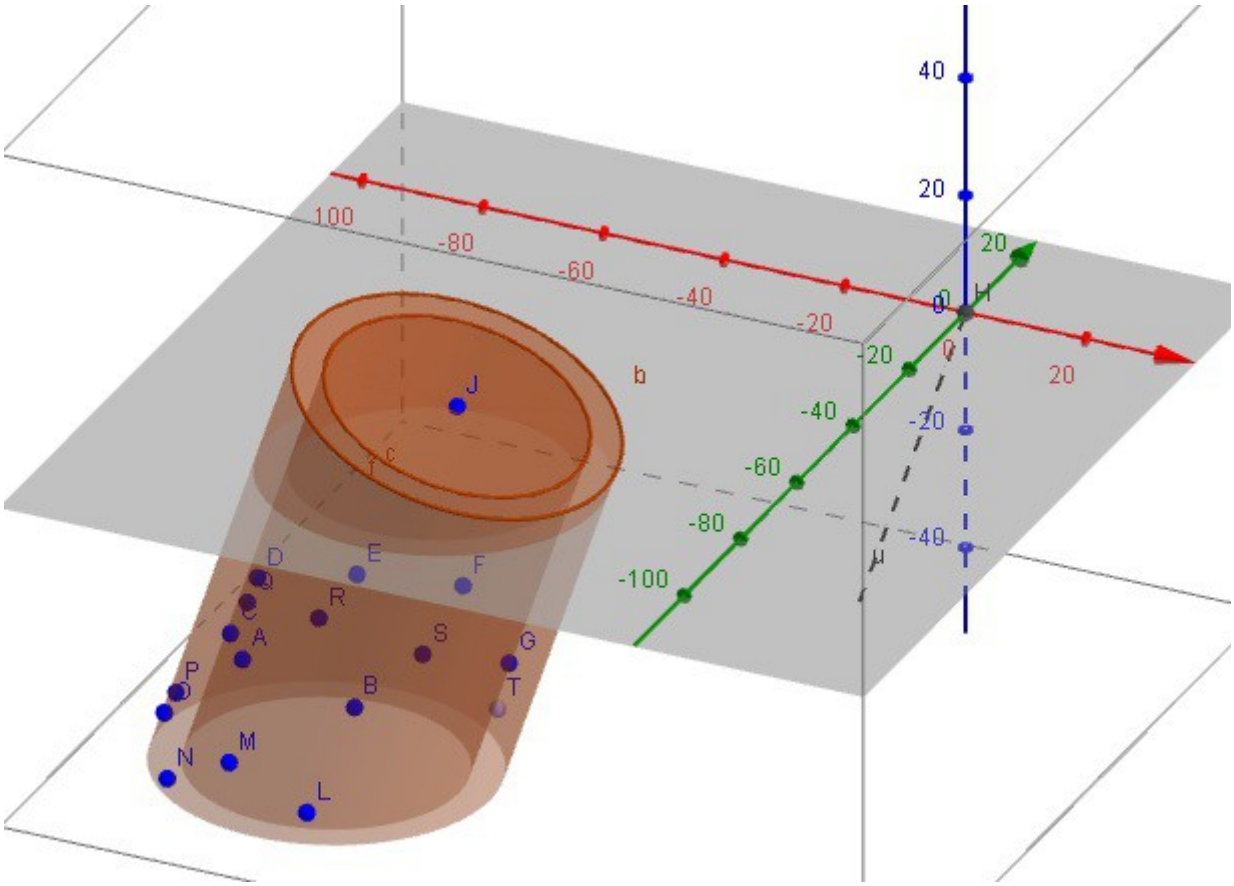


Ilustración 30: Representación de ambos cilindros

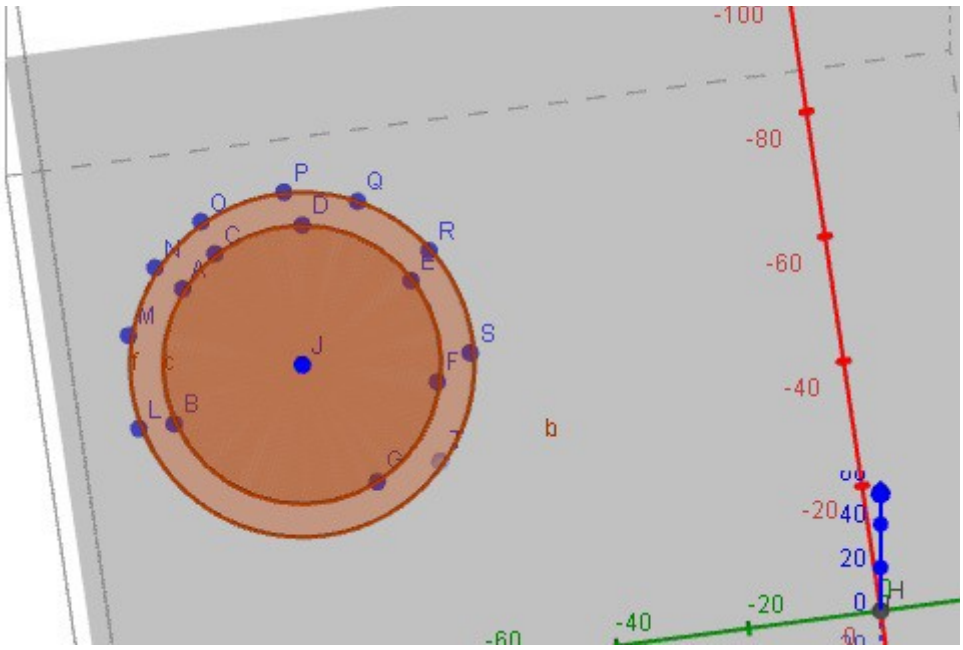


Ilustración 31: Vista superior

5.6.6 Cono vertical 1

Las dos piezas con forma cónica que se utilizaron para las pruebas aparecen en la ilustración 32.

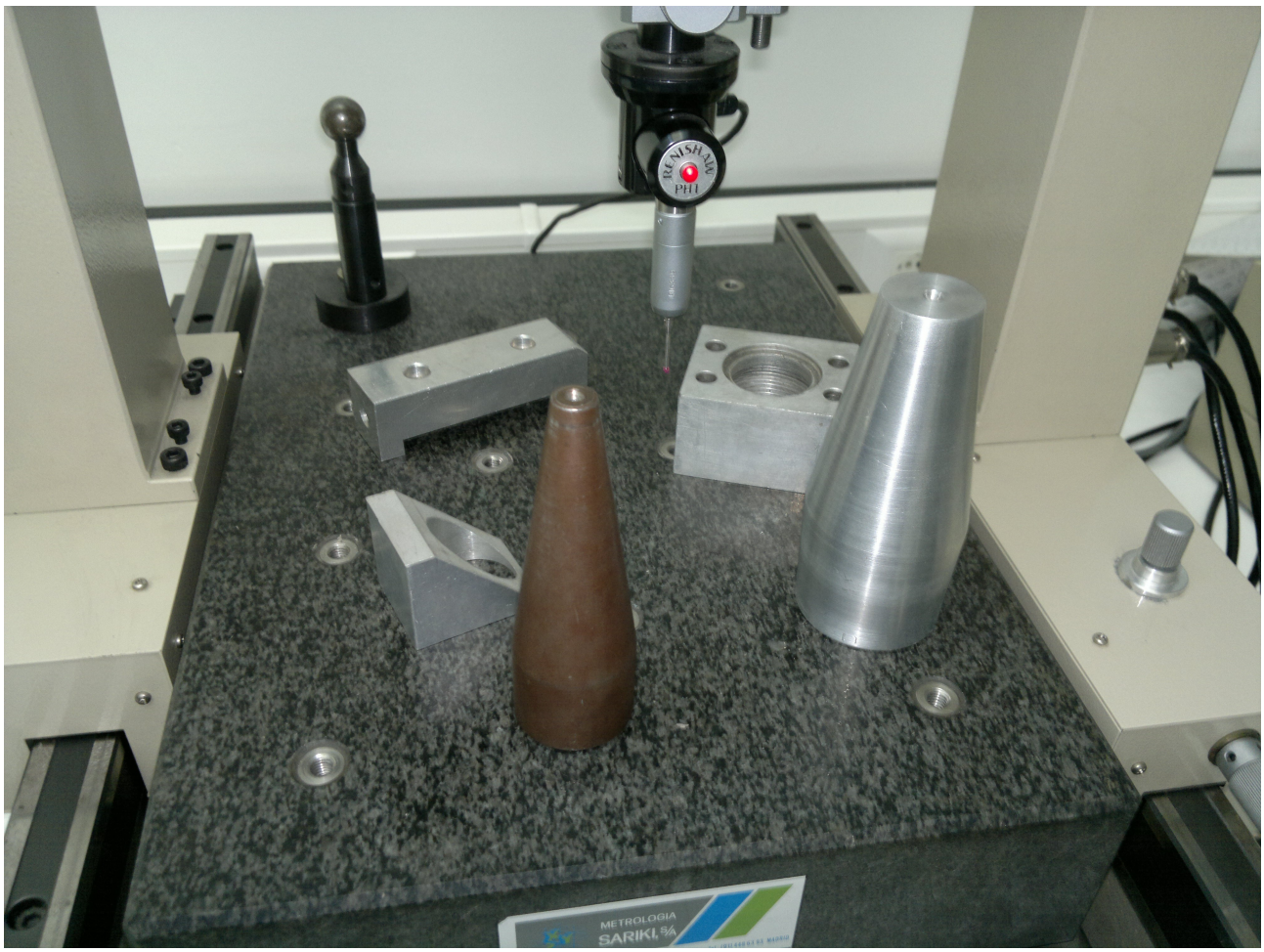


Ilustración 32: Cono de pruebas

Tomando los siguientes puntos

x	y	z
24,183	24,241	24,7
27,788	15,561	27,846
29,705	15,433	14,431
21,002	7,262	18,283
17,465	9,199	29,995
13,475	9,842	20,198
9,476	15,609	13,589
12,382	15,617	33,003
11,271	22,403	15,212
14,547	23,557	26,412
18,815	24,186	33,064
24,662	23,909	24,582

Ilustración 33: Nube de puntos del cono

El programa proporciona los siguientes parámetros

Ilustración 34: Resultado del cono

En la ilustración 35 se aprecia la recta de regresión

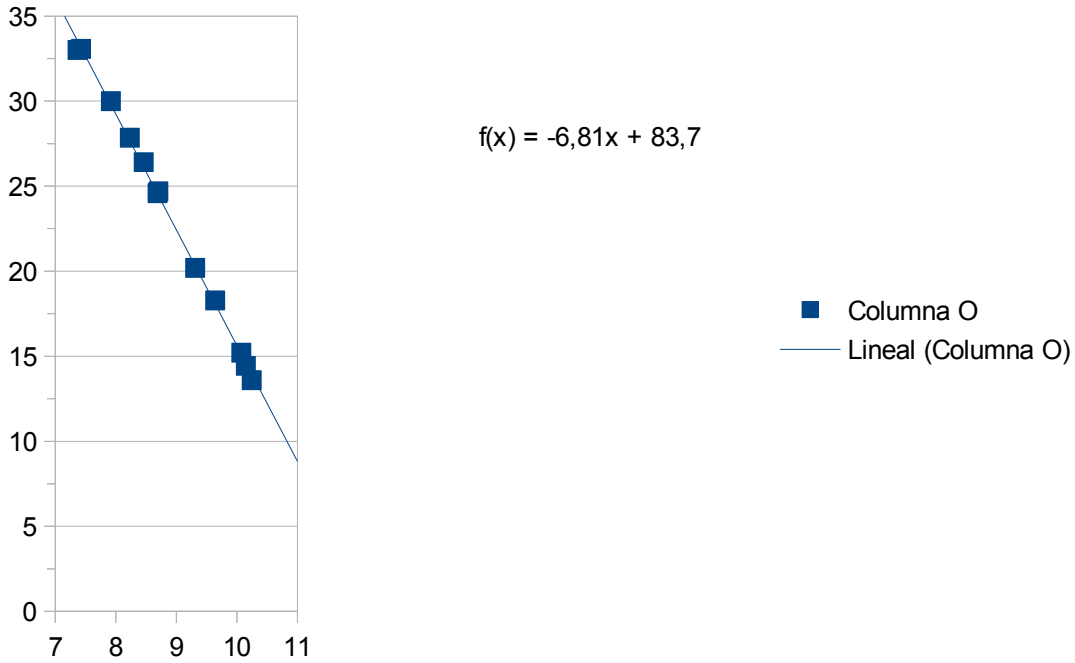


Ilustración 35: Recta de regresión del cono

A continuación mostramos el ajuste del cono a la nube de puntos

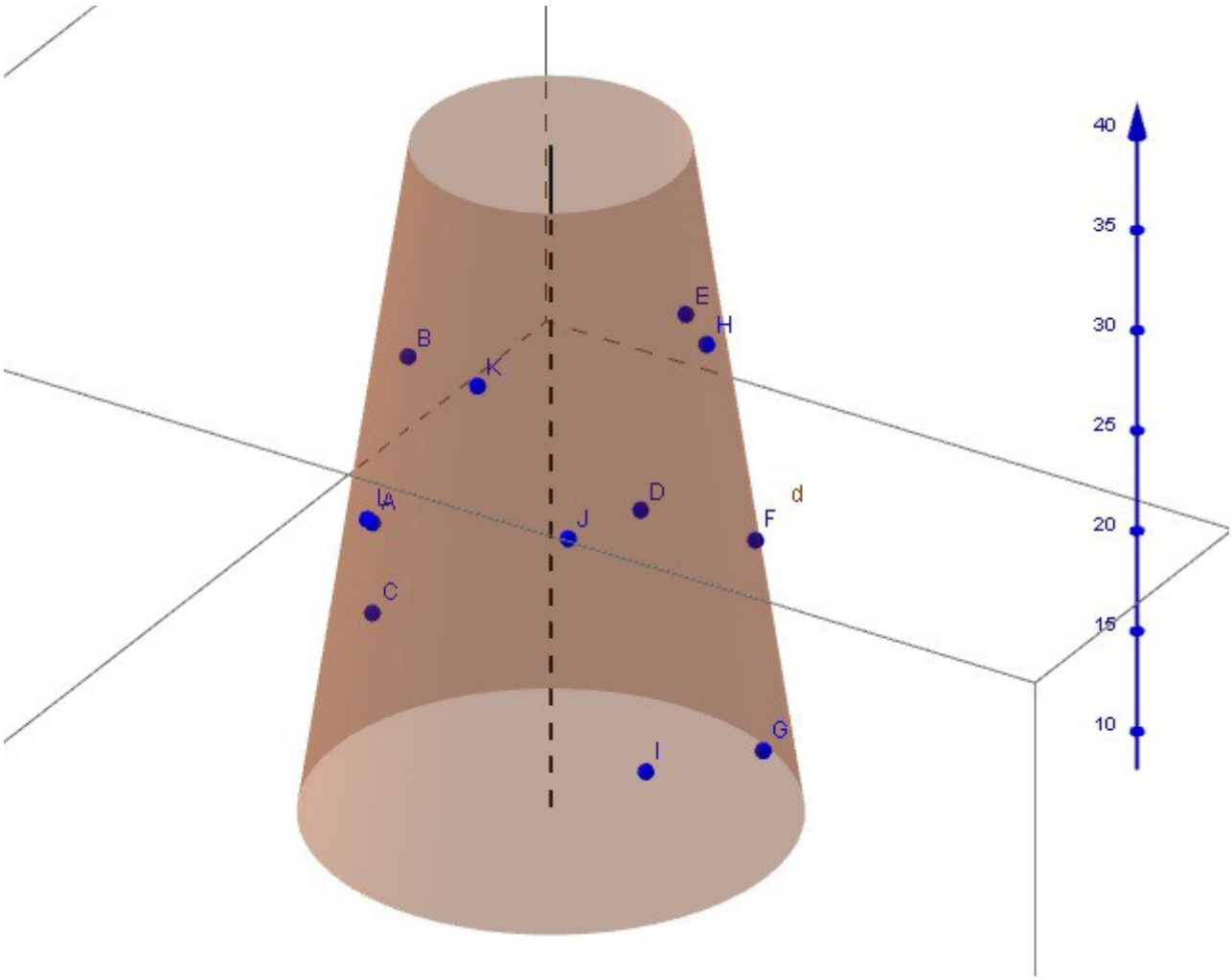


Ilustración 36: Representación del cono

5.6.7 Cono vertical 2

Para el segundo cono se tienen los puntos

x	y	z
49,484	19,825	47,58
51,611	19,802	33,301
42,034	33,709	46,062
22,258	32,083	48,739
14,805	20,04	36,495
21,953	4,589	33,159
36,006	1,003	32,134
44,532	5,128	34,206
39,775	4,442	46,732
46,801	7,776	36,339

Ilustración 37: Nube de puntos del cono

El programa proporciona los siguientes resultados

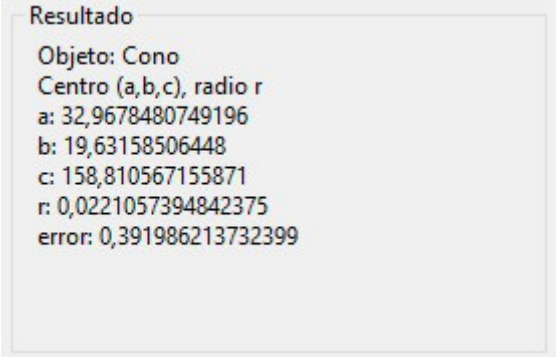


Ilustración 38: Resultado del cono

La superficie se ajusta a la nube de puntos como muestra la ilustración 39.

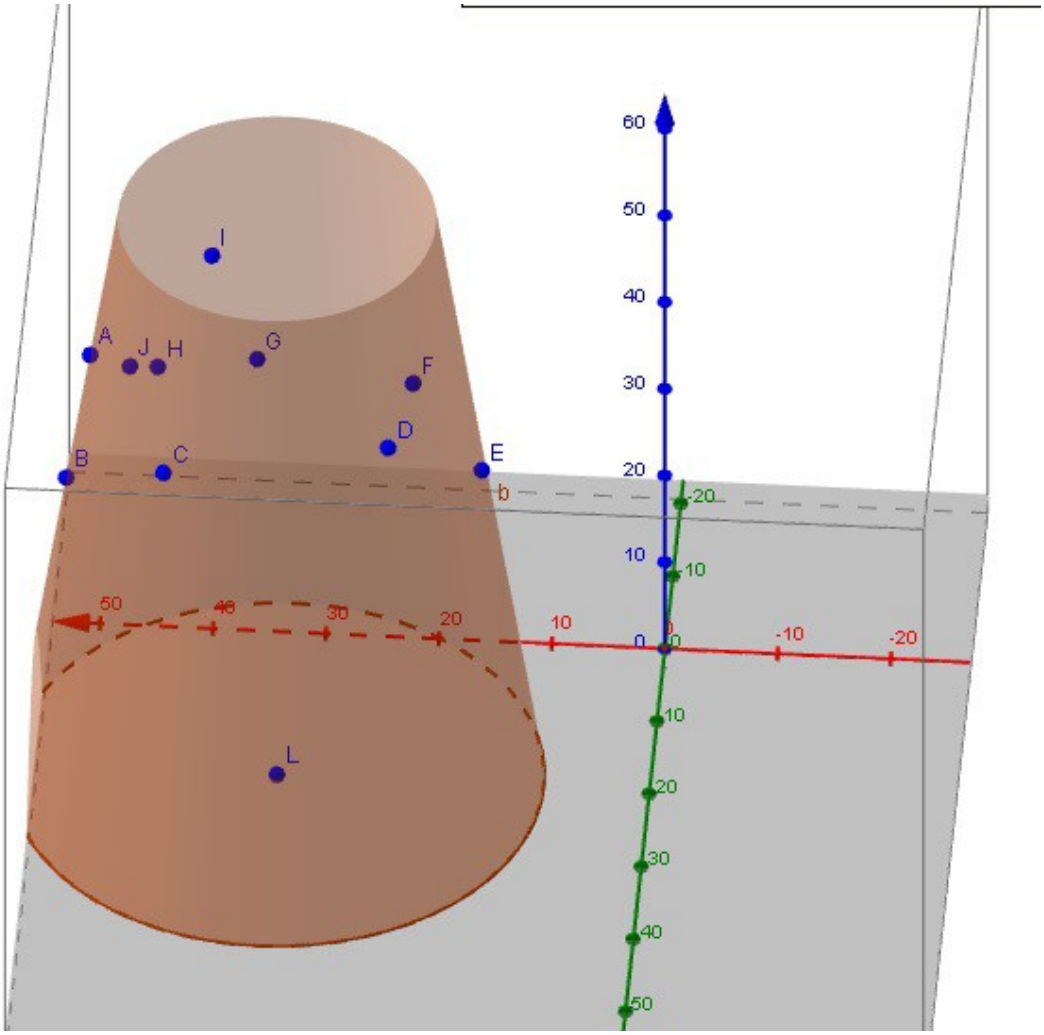


Ilustración 39: Representación del cono

5.6.8 Cono girado

La máquina proporciona los siguientes puntos

Cono girado		
x	y	z
-79,459	8,227	-11,776
-71,9	-30,714	41,382
-81,348	-20,378	29,273
-86,083	-7,712	10,665
-84,922	5,667	-12,579
-75,619	5,841	1,388
-43,845	-22,238	37
-49,405	-4,434	31,917
-64,587	1,924	25,625

Plano auxiliar		
x	y	z
-67,925	-16,045	48,62
-57,331	-7,552	47,256
-48,926	-15,596	45,31
-57,512	-27,559	46,113
-66,747	-27,869	47,705

Ilustración 40: Nube de puntos del cono girado

Al introducirlos en el programa se obtienen los siguientes resultados

Ilustración 41: Resultado del cono girado

Representando la nube de puntos junto con el cono obtenido tenemos las siguientes figuras.

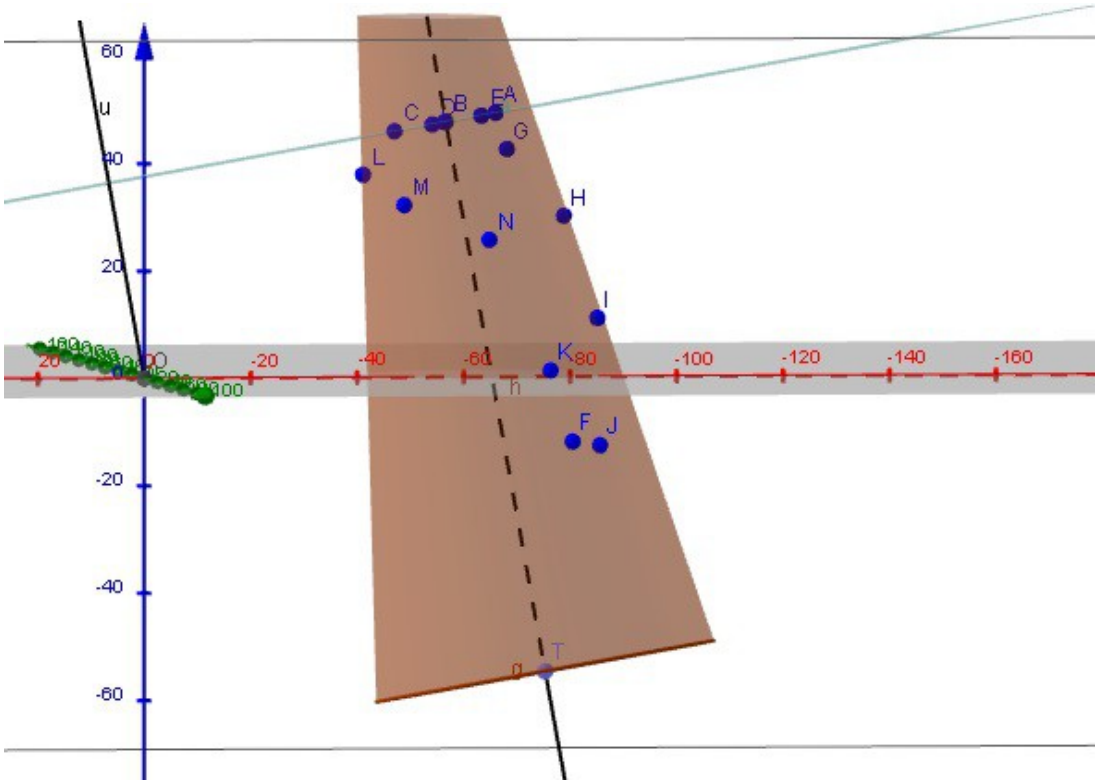


Ilustración 42: Representación del cono girado

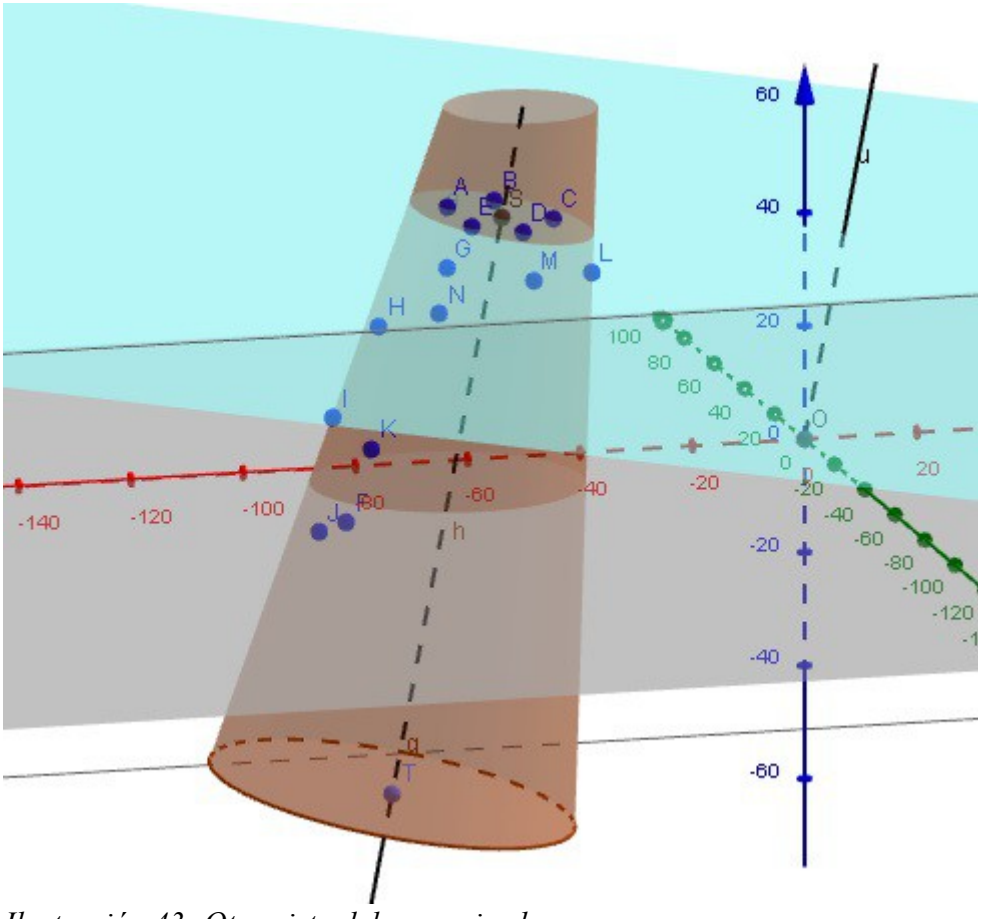


Ilustración 43: Otra vista del cono girado

5.7. Posibles mejoras

Hay dos aspectos que se pueden mejorar notablemente ya que para la obtención de los parámetros del cilindro o el cono es necesario introducir también el plano en el que se apoyan. Sería deseable no tener que hacer esto y para ello habría que proceder como en el caso de la esfera.

Otro punto a tener en cuenta es que el método de Newton-Raphson necesita una aproximación inicial a la solución para que el método converja. Dicha aproximación no parece fácil de obtener para el cilindro y el cono girados.

De hecho, para el cono recto (el que tiene el eje paralelo al eje *z*) tampoco se ha podido aplicar el método de Newton-Raphson y se ha tenido que calcular su ecuación exacta a partir de cierto número de puntos y posteriormente se han hecho las medias de los parámetros obtenidos al ir variando los puntos que tomaba en la lista de puntos totales.

Otro de los problemas que ha surgido con el cono ha sido el de determinar con exactitud la coordenada del vértice. Esta tarea es complicada, ya que cualquier variación mínima en la nube de puntos puede provocar un incremento de la tercera coordenada desproporcionado (basta pensar una nube de puntos de un cono con un semiángulo muy pequeño de forma que la superficie sea cercana a un cilindro). Por lo tanto, la tercera coordenada del vértice del cono no es un parámetro muy representativo, aunque si lo son su eje y su semiángulo.

Otro aspecto que también sería mejorable es el uso de funciones externas para la resolución de sistemas lineales, calculo de determinantes, distancias, vectores perpendiculares, etc.

5.8. Continuidad del proyecto

Hay varias líneas en las que el presente proyecto podría tener una continuidad.

En primer lugar, hay un proyecto paralelo para integrar el uso de la máquina de medición por coordenadas con un ordenador. En concreto lo que se pretende es que el ordenador lea la nube de puntos directamente de la máquina sin necesidad de introducirlos uno a uno manualmente. Para ello es necesario saber cómo la máquina transmite cada uno de los puntos al MICROPAK 100 y posiblemente sea necesario el diseño de un adaptador que permita emplear uno de los puertos del ordenador. Una vez que el ordenador pueda recibir los puntos, necesitará un pequeño programa para procesarlos y almacenarlos. Sería interesante que la nube de puntos leída se almacenase en el formato indicado anteriormente, aunque esto no supone un inconveniente real, ya que el formato de archivos de puntos utilizado por el programa *Descartes* es muy simple al ser archivos de texto con separadores.

En segundo lugar, cuando el proyecto anterior esté concluido, podría tratarse de integrar los dos programas en un único programa. Así, en lugar de leer la lista de puntos de un archivo como hace el programa Descartes, el propio programa leería los datos proporcionados por el palpador, lo que haría mucho más simple y rápido el uso de la máquina de medición por coordenadas.

En tercer lugar, podría tratar de mejorarse el programa Descartes tal y como se menciona en el apartado de Posibles mejoras, ya sea elaborando y perfeccionando las funciones matemáticas que utiliza o bien utilizando el método de Newton Raphson para el cilindro girado y el cono.

6. Conclusiones

Con el presente proyecto de desarrollo de un programa para la máquina de medición por coordenadas se ha alcanzado el principal objetivo que planteado que era desarrollar un programa funcional, fácil de usar y que permitiese aproximar las figuras básicas (esfera, cilindro, cono, plano, cilindro girado y cono girado) así como almacenar la lista de figuras obtenidas de una forma práctica y realizar cálculos de distancias y ángulos con ellas.

En la realización del programa han surgido una serie de dificultades que se han solventado y que me han hecho aprender de mis errores así como aprender a enfocar un problema de diversas estrategias, buscando aquella que mejor funcione.

A modo de resumen, me gustaría destacar las siguientes conclusiones a las que he llegado

Técnicas

- Dificultad para encontrar información precisa sobre los algoritmos que utilizan este tipo de programas.
- Existencia de variedad de algoritmos en algunos casos (mínimos cuadrados, promedios, etc.) sin que parezca que este claro cuál es el mejor de ellos.
- En el caso del cono, cuando el semiángulo es pequeño, la posición del vértice es muy sensible a pequeños errores de medición, y por lo tanto no debería considerarse relevante. Sí que son relevantes el eje y el semiángulo.
- El cono y el cilindro girados los he resuelto con ayuda de un plano auxiliar, ya que no he podido encontrar información para hacerlo sin usar dicho plano.

Personales

El desarrollo del proyecto me ha supuesto muchas horas de investigación. He tenido que mejorar mis conocimientos matemáticos y mis conocimientos sobre programación. En concreto, durante la elaboración del proyecto he ampliado considerablemente mis conocimientos y mi entendimiento sobre los siguientes temas

- Geometría vectorial en el espacio.
- Resolución de sistemas por métodos numéricos.
- Programación en un entorno orientado a ventanas.
- Uso de herramientas matemáticas para representación de objetos geométricos.

En definitiva, con este proyecto no sólo he mejorado mis habilidades matemáticas y computacionales sino que también he aprendido a buscar distintas estrategias para resolver problemas reales, a perseverar en el trabajo y a organizar mis tareas.

7. Bibliografía

[1] Altemir, J. M., Esteban, E., Moya, A. y Torres, F. *Diseño y construcción de una máquina de medida de coordenadas asistida por ordenador para una fundición.*

[2] Apostol, T. M. *Calculus, vol 1.*

[3] Apostol, T. M. *Calculus, vol 2.*

[4] Cruces, S. El método de mínimos cuadrados. http://personal.us.es/sergio/PDocente/lectura.pdf

[5] Galván, C. Algoritmos de ajuste para máquinas de medición por coordenadas.

[6] Hernández, E. *Álgebra y Geometría.*

[7] Lazarus-ide. http://www.lazarus-ide.org/

[8] Lazarus Tutoriales. http://wiki.freepascal.org/Lazarus_Tutorial/es

[9] Moreno, C. *Introducción al cálculo numérico.*

[10] Muñoz, R. Introducción a la metrología.

[11] Tutorial de Lazarus. http://lazarustutorials.blogspot.com.es/2013/04/save-dialog.html

[12] Wikipedia. Método de Newton-Raphson.

[13] Wikipedia. El método de mínimos cuadrados. http://es.wikipedia.org/wiki/M%C3%ADnimos_cuadrados

[14] Wiki.freepascal.org Diálogo de guardado. http://wiki.freepascal.org/Howto_Use_TSaveDialog

[15] El control TstringGrid. http://www.formauri.es/arrobamasmas/Cursos/index.php?apdo=0402&curso=4_02_03&cap=3

Anexo I. Manual de usuario

Anexo II. Manual de Micropak

Anexo III. Código

Proyecto fin de carrera

Ingeniería Técnica Mecánica

ANEXO I:

MANUAL DE USUARIO

Autor: José Tovío Ciércoles

Director del proyecto: Mª Rosario González Pedraza

Especialidad: Ingeniería Técnica Mecánica

Fecha: 21 de mayo de 2015

Convocatoria: junio de 2015

Descartes 1.0

Manual de usuario

José Tovío Ciércoles

1 de febrero de 2015

Índice de contenido

1. Introducción

Una máquina de medición por coordenadas es un instrumento de medida capaz de determinar las medidas y la forma de un objeto sobre el que se aplique. Para ello dispone de un palpador que se posa sobre la superficie de la pieza a medir. El palpador se mueve mediante tres ejes perpendiculares y cuando esta parado estos ejes proporcionan las coordenadas cartesianas X, Y, Z del centro del palpador. Tomando una muestra de estas medidas (supondremos que no más de 20) e indicando el tipo de superficie que está siendo medida, la máquina determina sus principales parámetros.

El programa calcula las superficies definidas por el palpador: planos, cilindros, conos y esferas. Recibe como entrada una lista de puntos (20 como máximo) y calcula los parámetros que definen la superficie correspondiente. El usuario del programa debe cargar la lista de puntos, indicar la posición del palpador respecto del objeto e indicar el tipo de objeto.

Los elementos geométricos se almacenan como una fila de una matriz que consta de los parámetros que los definen. Para ello hemos considerado los parámetros que se indican a continuación.

Superficies determinadas directamente por el palpador:

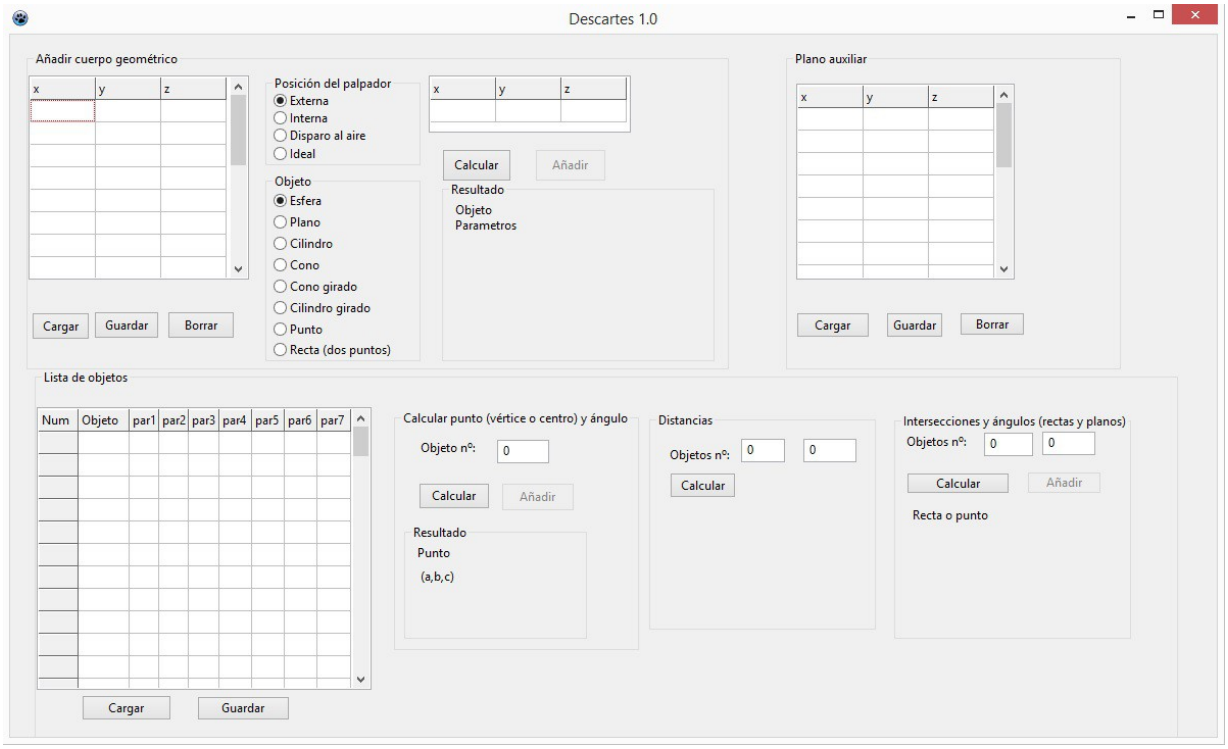
- **Plano**: 6 parámetros: un punto del plano *(a,b,c)* y un vector normal unitario *(u,v,w)*.
- **Esfera**: 4 parámetros: el centro *(a,b,c)* y el radio *r*.
- **Cilindro recto**: 3 parámetros: el centro *(a,b)* y el radio *r*.
- **Cono recto**: 4 parámetros: el vértice *(a,b,c)* y el parámetro *r* en la ecuación $(x-a)^2+(y-b)^2=r(z-c)^2$. Geométricamente este parámetro es el cuadrado de la tangente del semiángulo ($\tan a=\sqrt{r}$).
- **Cilindro girado**: 7 parámetros: un punto del eje *(a,b,c)*, un vector dirección *(u,v,w)* y el radio *r*.
- **Cono girado**: 7 parámetros: un punto del eje *(a,b,c)*, un vector dirección *(u,v,w)* y el cuadrado de la tangente del semiángulo.

Elementos derivados mediante cálculos:

- **Punto**: 3 parámetros: el propio punto *(a,b,c)*.
- **Recta**: 6 parámetros: un punto *(a,b,c)* y un vector dirección *(u,v,w)*.

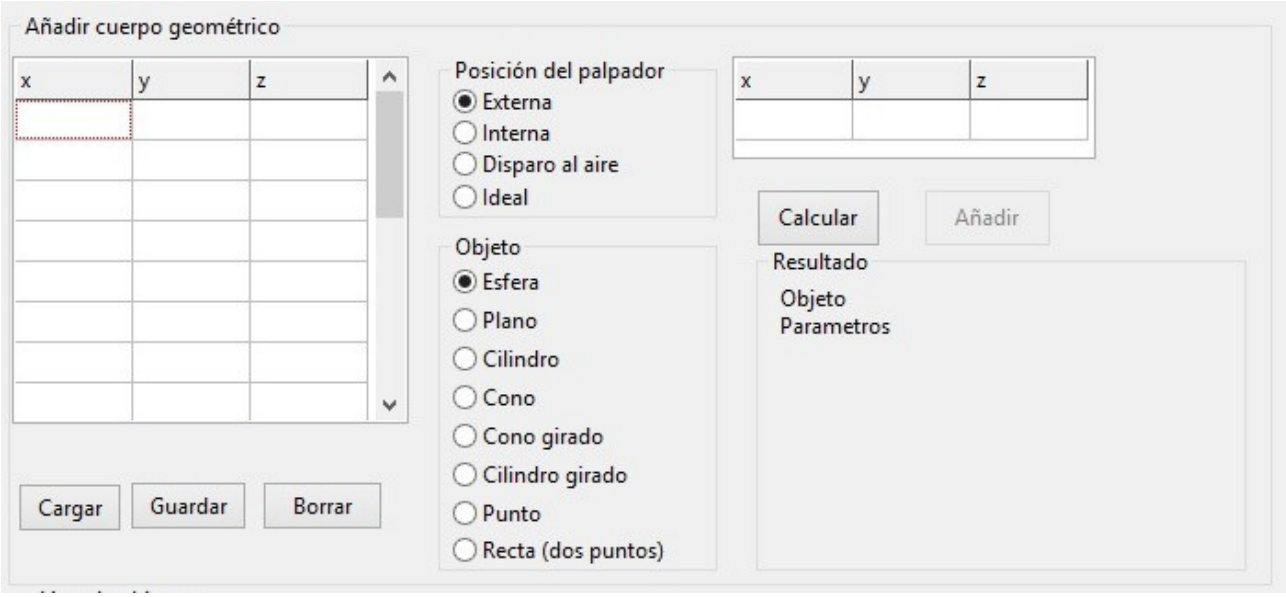
A partir de los objetos almacenados el programa calcula intersecciones, ángulos y distancias. También permite guardar una lista de objetos para poder recuperarla en una sesión posterior.

2. Descripción de la pantalla principal

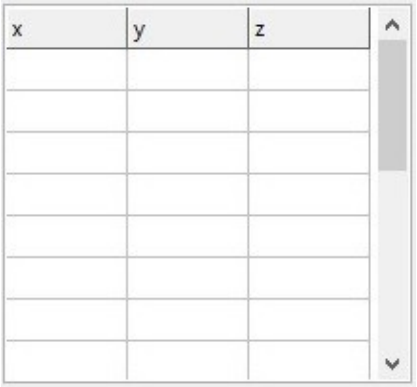


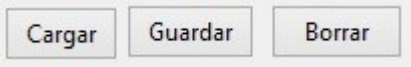
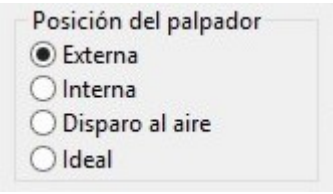
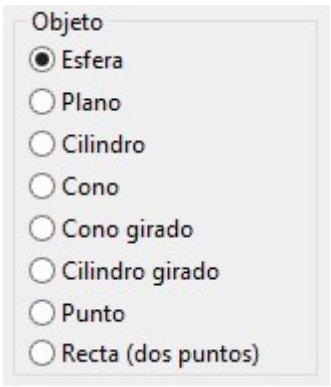
La pantalla principal de Descartes 1.0 aparece dividida en tres zonas.
Caja Añadir cuerpo geométrico. Utilizamos esta zona para introducir la lista de puntos, seleccionar el tipo de objeto y la posición relativa del palpador y realizar el cálculo de los parámetros de la superficie, así como para añadir los resultados calculados a la lista de objetos.
Caja Plano auxiliar. Se utiliza esta zona para introducir la lista de puntos del plano auxiliar cuando sea necesario.
Caja Lista de objetos. En esta zona se muestra la lista de objetos que se han ido calculando y permite realizar operaciones y cálculos entre ellos.

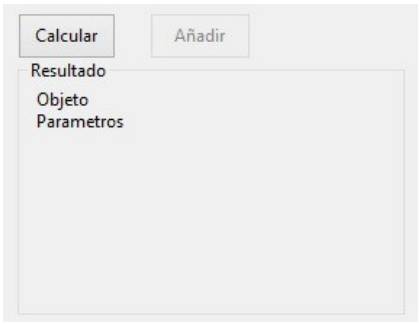
3. Zona Añadir cuerpo geométrico



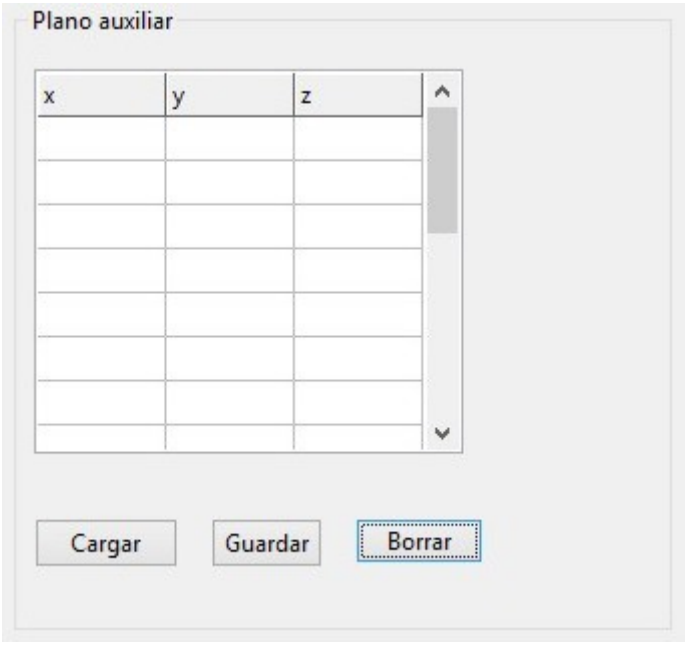
Esta zona consta de varios elementos que se describen brevemente en la siguiente tabla

	La caja se utiliza para introducir la lista de puntos. Se puede hacer manualmente introduciendo los puntos uno a uno o bien se puede pegar una lista de puntos pulsando Ctrl+v que se haya copiado previamente desde una tabla de una hoja de cálculo. Al introducir los puntos, los caracteres están limitados a las cifras del 0 al 9, el signo menos y la coma como separador decimal. Si

	se intenta introducir cualquier otro carácter el programa emitirá un pitido de error.
	Los botones de debajo de la caja permiten realizar las operaciones que indican: Cargar: carga una lista de puntos desde un archivo. El archivo que se carga tiene que haber sido guardado previamente desde el propio programa. Esto es así porque esta guardado en modo texto y utiliza un separador para delimitar los campos. Guardar: guarda la lista de puntos en un archivo de texto. Borrar: borra todas las casillas de la caja de texto.
	El palpador que realiza las mediciones tiene un radio determinado que influye en los cálculos posteriores. Por eso es necesario saber que cuál es la posición del palpador respecto del objeto medido. Hay cuatro opciones: **Externa**: el palpador está fuera de la esfera, cilindro o cono. **Interna**: el palpador está dentro. **Disparo al aire**: en lugar de indicar la posición del palpador directamente, introducimos un punto más en las casillas que a tal efecto se encuentran a la derecha. Es la única opción disponible para el cálculo de planos. En caso de no especificar esta poción para el plano, se calcula el plano ideal, es decir, no se tiene en cuenta el radio del palpador. **Ideal**: no tiene en cuenta el radio del palpador. Esta posición era necesaria para realizar pruebas con el programa.
	En la lista de objetos podemos elegir el tipo de objeto que vamos a determinar. Hay que tener en cuenta ciertas condiciones de funcionamiento: Cilindro y cono: el programa considera que estos objetos tienen el eje en la dirección del eje z. Cono girado y cilindro girado: es necesario introducir un plano auxiliar que se supone que es perpendicular a la dirección del eje para que el programa pueda hacer los cálculos. Punto: únicamente lee el primer punto de la lista, para poder añadirlo a la lista de objetos.

	Recta: únicamente tiene en cuenta los dos primeros puntos de la lista. De nuevo esto es así para poder añadir de forma sencilla rectas a la lista de objetos.
	En esta caja se introduce el disparo al aire realizado por el palpador para poder determinar la posición de éste respecto del objeto. Tiene que estar marcada la casilla “Disparo al aire” de la izquierda para que realmente se usen estos valores. El disparo al aire es la única opción posible para el cálculo de planos.
	Una vez introducidos la lista de puntos, el tipo de objeto y la posición del palpador, podemos realizar el cálculo de los parámetros de la superficie pulsando sobre el botón calcular. El resultado se muestra justo debajo del botón. Además podremos añadir el objeto calculado a la lista de objetos pulsando sobre el botón Añadir. En caso de no hacerlo el objeto no es añadido a la lista y se perdería al realizar un nuevo cálculo.

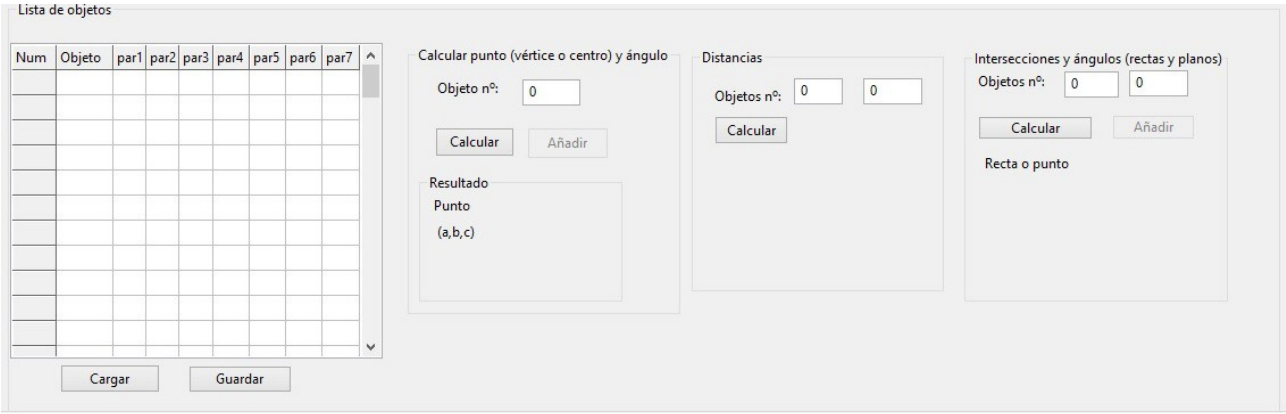
4. Zona Plano auxiliar



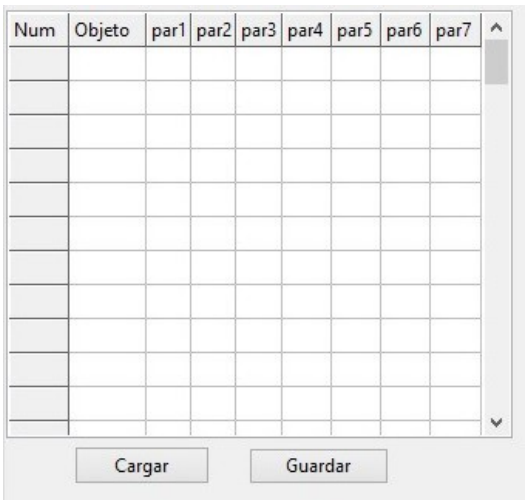
Esta zona es muy parecida a la anterior aunque consta de menos elementos. En la caja que aparece

se introduce una lista de puntos del plano auxiliar necesario para calcular un cono o cilindro cuyo eje no es paralelo al eje z. De nuevo aparecen tres botones para manipular la lista de puntos que se introduce y al igual que antes esta lista puede pegarse desde la hoja de cálculo.
Hay que insistir que esta zona únicamente se utiliza cuando el objeto que se calcula es un cono girado o un cilindro girado. Si se esta calculando cualquier otro objeto, la lista de puntos que aparezca aquí no tendrá ningún efecto.

5. Zona Lista de objetos



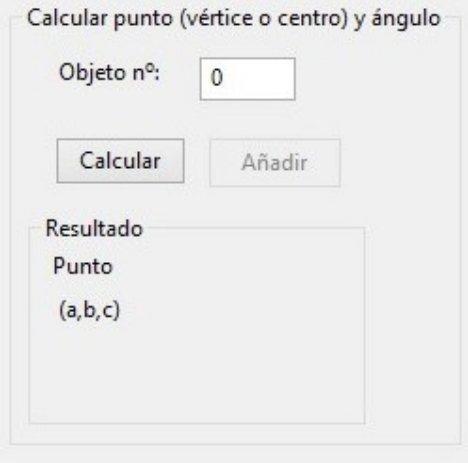
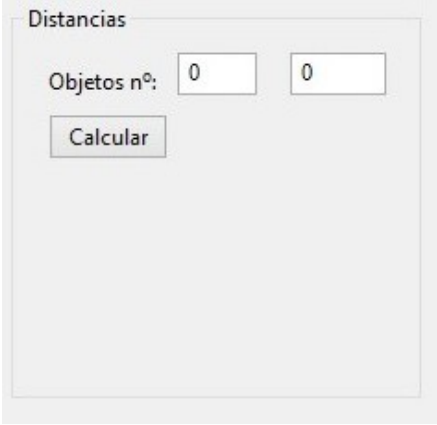
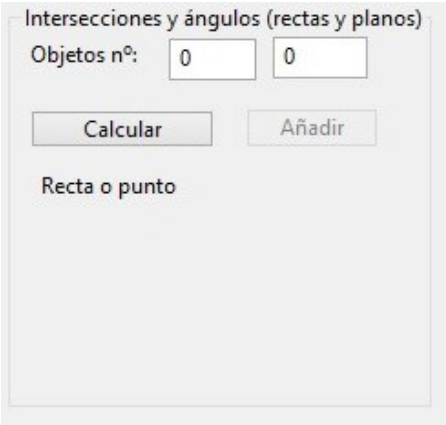
Esta zona aparece divida en cuatro partes que se describen a continuación.



La lista de objetos que están en memoria aparece en este cuadro. Cada objeto está numerado para poder realizar las operaciones correspondientes entre ellos. Aparecen dos botones debajo de la lista

- Guardar: guarda la lista de objetos como archivo de texto.
- Cargar: carga una lista previamente guardada.

El tipo de objeto viene determinado por el tipo de objeto marcado a la hora de realizar el cálculo y los parámetros dependerán del tipo del msimo.
Todo esto se detalla más abajo.

	En esta caja podemos calcular un punto y un ángulo. Su comportamiento depende del tipo de objeto introducido. Así, según el objeto se tiene: Esfera: muestra el centro Cono: Muestra el vértice y el semiángulo. El botón Añadir permite añadir el punto a la lista de objetos.
	Permite calcular la distancia entre dos objetos distintos siempre que esta distancia tenga sentido. El resultado se muestra en una ventana emergente.
	Permite calcular la recta de intersección entre dos planos y el ángulo que forman o si se introduce una recta y un plano entonces muestras el punto intersección y el ángulo que forman.

6. La lista de objetos

En la lista de objetos pueden aparecer los objetos que hemos mencionado al comienzo de este manual. Los parámetros que determinan cada objeto se dan en el mismo orden que aparecen.

Proyecto fin de carrera

Ingeniería Técnica Mecánica

ANEXO II:

MANUAL DE MICROPAK

Autor: José Tovío Ciércoles

Director del proyecto: Mª Rosario González Pedraza

Especialidad: Ingeniería Técnica Mecánica

Fecha: 21 de mayo de 2015

Convocatoria: junio de 2015

ANEXO I

1.1. PUESTA EN MARCHA

Se conecta la máquina con el interruptor rojo situado en la superficie lateral derecha del MICROPAK.
La máquina realiza un autochequeo. Al finalizar esta función avisará con una señal sonora. Se muestra en la pantalla la siguiente información:

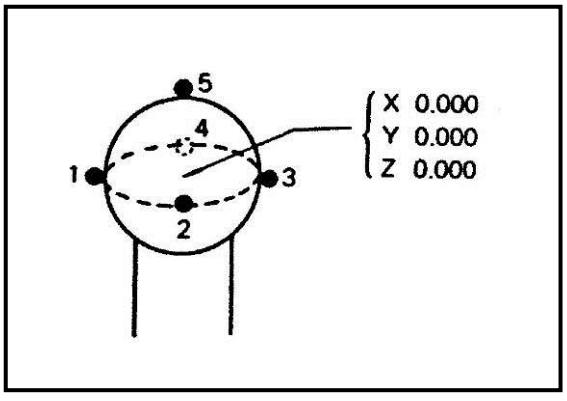
X	
Y	
Z	21.00100
S	P - 100 01

Donde:
2**1**.00100 indica lectura en mm. (2**2** supondría lectura en pulgadas)
21.**00100** indica resolución de 0.001 mm.
P - 100 **01** indica versión primera del programa

Cuando se encuentran defectos en el autochequeo el MICROPAK muestra su código de errores en la pantalla de operaciones y detiene su funcionamiento.
El usuario de la máquina podrá posicionar como desee el palpador o cambiarlo por otro. Cada vez que se altere su postura será necesario almacenar en memoria el origen de la máquina para compensar la desviación del centro de la esfera de rubí del palpador. Esto se podrá hacer midiendo una bola patrón para colocar el origen del sistema de coordenadas de la máquina en su centro.

1.- Pulsar la tecla E. Aparece en pantalla el diámetro de la bola patrón. En este caso, dicho diámetro es 2 mm. Si se utiliza otro patrón, el valor será diferente y habrá que cambiarlo. Para más información, consultar "Operation Manual".
2.- Pulsar la tecla 3D-PROBE.
3.- Pulsar una de las teclas de instrucción de medición (MEASURE) El LED de la tecla DATA-IN debe estar encendido.
4.- Pulsar la tecla OP. Aparece en pantalla:

X : 0.000
Y : 0.000
Z : 0.000
S : 5

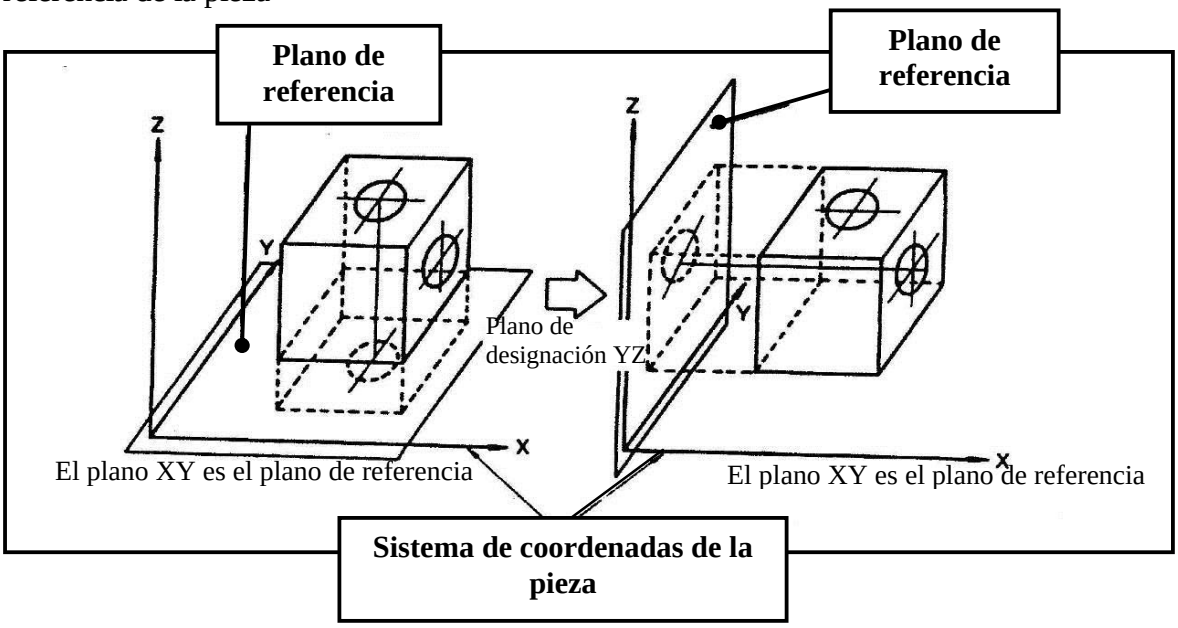


Siendo S el número de puntos que necesita la máquina para realizar la operación.
5.- Introducir los cincos puntos mediante el palpador en cualquier orden. Es recomendable introducir cuatro puntos aproximadamente a lo largo del ecuador de la esfera y otro en el polo, tal como se indica en la figura,

para minimizar el error. El origen de la máquina queda definido en el centro de la esfera. La posición del centro de la esfera de rubí del palpador se compensa automáticamente.

A. DESIGNACIÓN DEL PLANO DE REFERENCIA

Esta función se emplea para designar el plano de referencia sobre el que se proyectan los puntos introducidos para medir determinadas geometrías. Se utilizará siempre que se cambie de orientación el palpador y se realizará inmediatamente antes de la alineación del plano.
Si no se hace designación expresa, el plano X-Y se designa automáticamente como plano de referencia de la pieza



1. Pulsar la tecla SHIFT
2. Pulsar la tecla P-DESG
3. En función del plano que se quiera designar se pulsará:
Plano X-Y: pulsar 4
Plano Y-Z: pulsar 5
Plano Z-X: pulsar 6
4. Pulsar E.

Ejemplo:

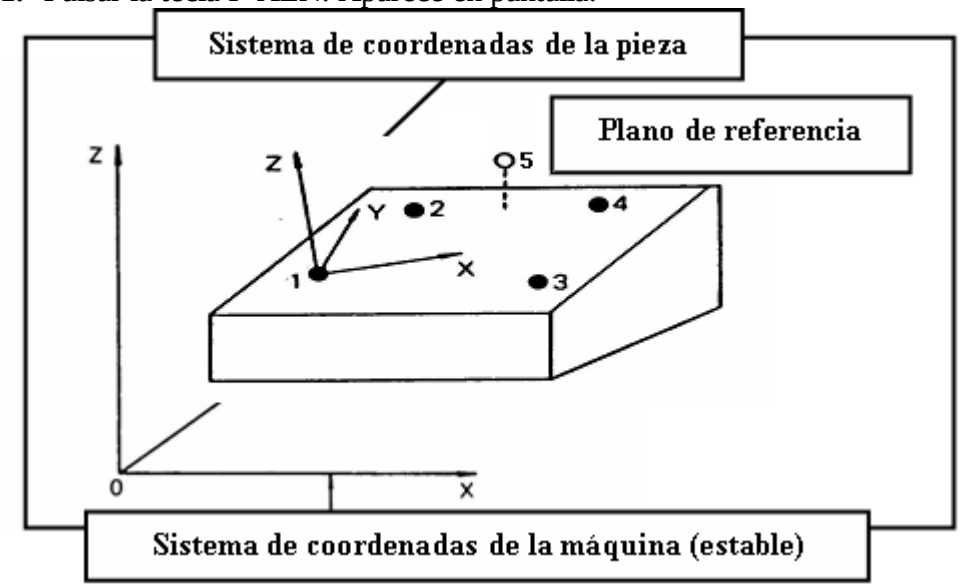
1. SHIFT + P-DESG + 5 + E
2. Definir el Y-Z como plano de referencia.

El uso de la función de designación de plano hace posible tomar mediciones 3D de una pieza cuando se cumplan las condiciones siguientes:

1. Un sistema de coordenadas de la pieza se puede establecer sobre una cara individual de la misma
2. La cara a medir debe ser paralela o perpendicular al sistema de coordenadas de la pieza establecido
3. La característica a medir de la pieza debe ser perpendicular a la cara.

B. ALINEACIÓN DE PLANO

Esta función se utiliza para determinar los planos de referencia de la pieza a medir.
1.- Pulsar la tecla P-ALN. Aparece en pantalla:



2.- Introducir 4 puntos del plano lo más distantes posibles, para evitar errores, y un dummy (el primer punto introducido es el que la máquina toma como origen de la pieza).
Aparece impreso:

:P-ALN OMG = 0.0213 (p.e.)
Siendo la lectura: 0 Grados, 02 Minutos, 13 Segundos

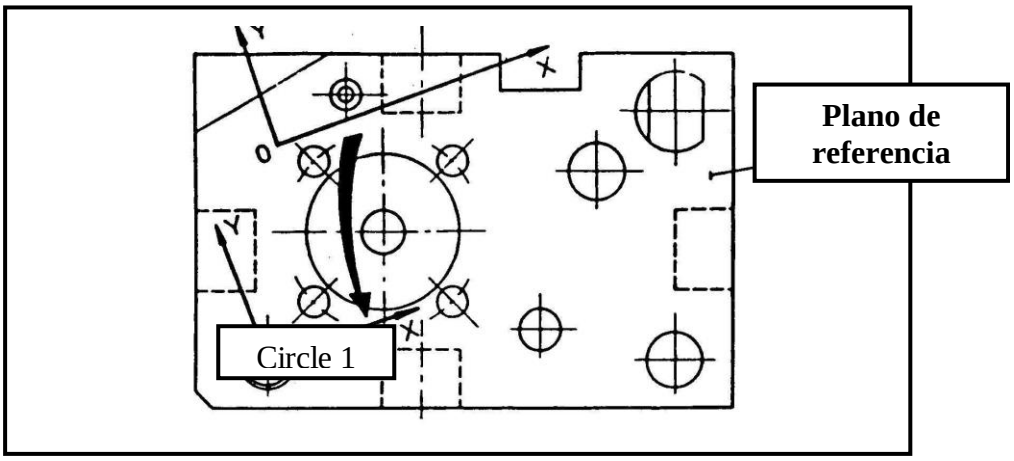
OMG: Inclinación de una pieza respecto del sistema de referencia de la máquina.

Para realizar cualquier tipo de medida habrá que definir un plano de referencia para la misma, cambiando dicho plano en función de la medida que se vaya a realizar.

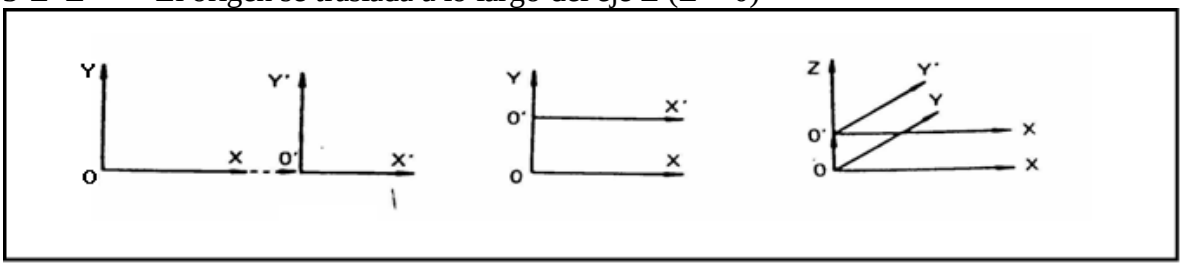
C. TRASLACIÓN DEL ORIGEN

Esta función se utiliza para efectuar una traslación del origen del sistema de coordenadas de la pieza en el plano de referencia definido

1. Se realiza una operación que contenga el punto al que se quiere trasladar el origen. Por ejemplo, si queremos que el nuevo origen sea el centro de un círculo, introduciremos la orden CIRC.
2. Pulsar ORG.
3. Pulsar O-PLN y E.
4. En la impresión del papel quedará reflejado, siguiendo con el ejemplo anterior, el diámetro del círculo.

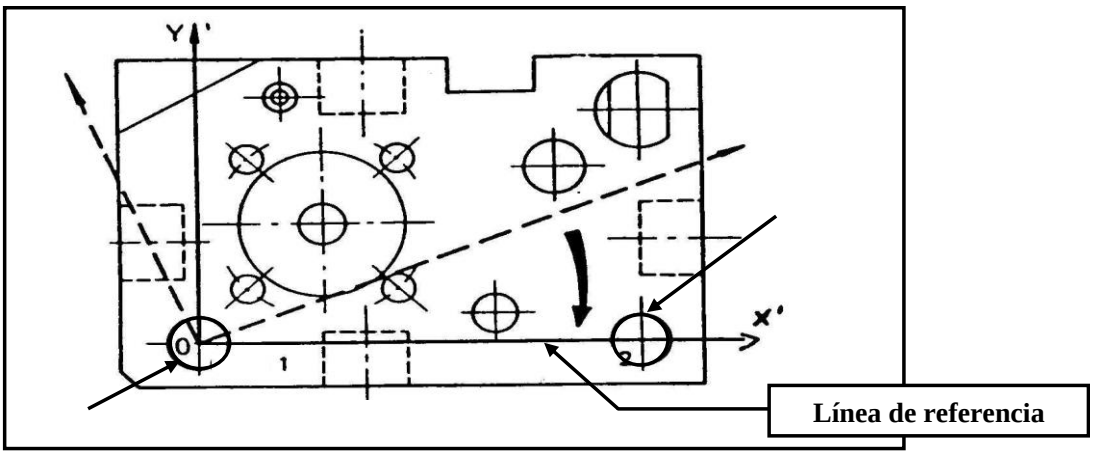


Nota:
O-PLN E ==> El origen se traslada al punto medido (X´= Y´= 0)
1-X E ==> El origen se traslada a lo largo del eje X (X´= 0)
2-Y E ==> El origen se traslada a lo largo del eje Y (Y´= 0)
3-Z E ==> El origen se traslada a lo largo del eje Z (Z´= 0)



D. ALINEAMIENTO DE EJES

Esta función se usa para determinar la dirección del eje de referencia (eje X en el caso del plano XY) usando el punto recién medido en el plano de referencia. Ejemplo: después de haber trasladado el origen a un punto del plano de referencia, medir un cilindro sobre dicho plano para alinear el eje X con el centro de dicho círculo.



1. Medir una característica de la pieza que determine un punto que defina el eje que se desea establecer como referencia

2. Pulsar la tecla A-ALN en la columna REF DETERM

Aparece en la pantalla P para indicar que se está ejecutando el proceso

3. Se imprime el siguiente mensaje:

:CIRC 2
D=nn.nnn
:A-ALN

E. ALINEACIÓN DE LOS EJES OFFSET (COMPENSADOS)

Esta función se utiliza para alinear (girar) el sistema de coordenadas de la pieza de forma que el punto anterior medido será localizado a partir de los ejes alineados por la introducción de sus coordenadas en el sistema actual. El origen no se desplaza por este proceso.

Se puede efectuar una compensación multiejes, cuando los valores de las dos coordenadas de ambos ejes del plano de referencia se conocen.

Una vez medida una característica de la pieza que determine el punto con el que se realizará la alineación:

1.- Pulsar las teclas SHIFT y OFFSET. Aparece en pantalla:

S: 1 0.000; donde 1 alude al eje X

2.- Introducir el valor de compensación para el eje X. Terminar pulsando la tecla E. Aparece en pantalla:

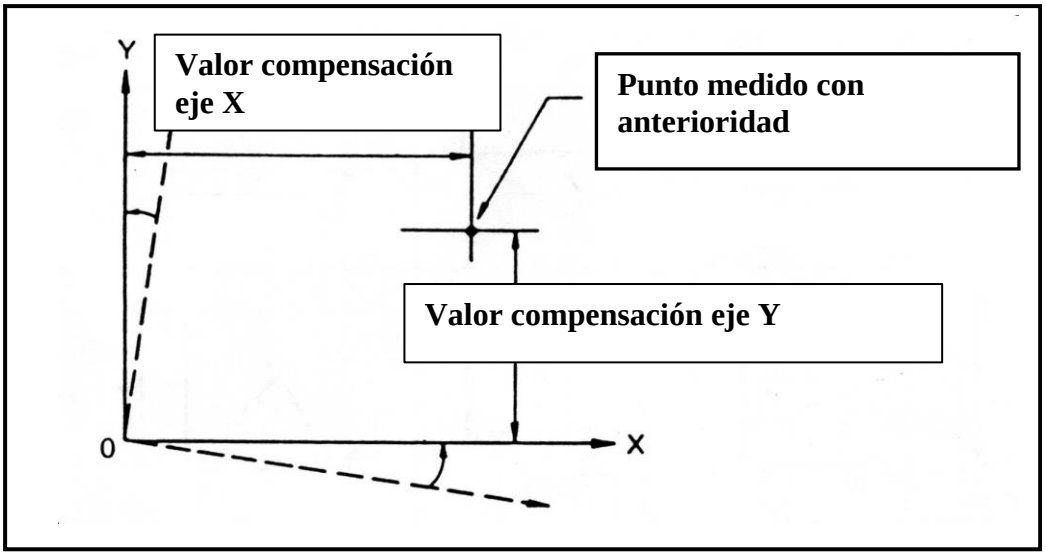
S: 2 0.000; donde 2 alude al eje Y

3. Introducir el valor de compensación para el eje Y. Terminar pulsando la tecla E. El proceso está completo y se imprime el siguiente mensaje:

: PT

1

:OFFSET X = n.nnn Y = n.nnn



Si sólo se conoce un valor de compensación en un plano se introduce de la forma expresada y, en el caso del otro eje, se pulsan las teclas 0 y E ó -, 0 y E para indicar si el valor desconocido es positivo o negativo.

F. ROTACIÓN DEL SISTEMA DE COORDENADAS

Esta función se utiliza para girar el sistema de coordenadas de la pieza alrededor del eje perpendicular al plano de referencia un ángulo dado.

1.- Pulsar SHIFT y ROTA. En la pantalla aparecerá:

S 0.0000

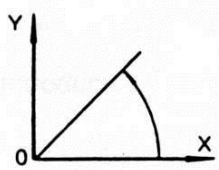
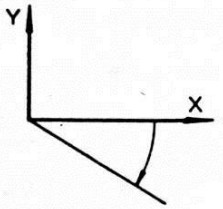
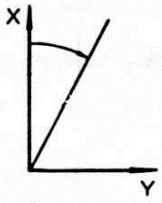
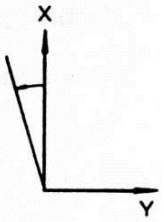
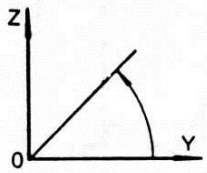
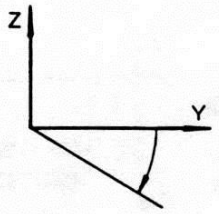
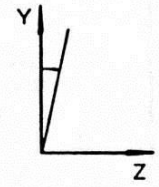
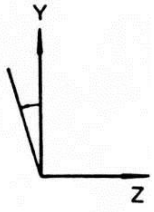
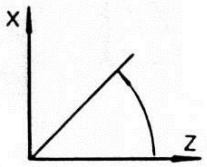
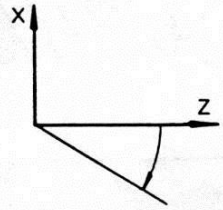
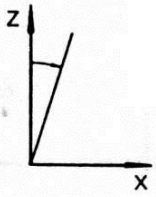
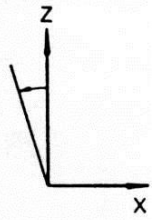
2.- Introducir el ángulo de giro que se desee. La gama permitida para el ángulo de rotación es de $-360º < t < 360º$. En pantalla aparecerá:

S n.nnnn (ángulo de rotación introducido)

3.- Pulsar E. Aparece impreso:

:ROTA A = n.nnnn

El sentido de giro positivo es contrario a las agujas del reloj.

A / Plano de referencia	Valor positivo	Valor negativo	Valor positivo	Valor negativo
X Y				
Y Z				
Z X				

G. TIPOS DE MEDICIONES

Esta función se utiliza para establecer uno de los tres tipos más frecuentes de sistema de coordenadas de la pieza utilizando una única operación de tecla

MODELO A: Los ejes se sitúan en una esquina de la pieza y el eje de referencia coincide con la cara lateral de la pieza.

1.-Pulsar la tecla A para seleccionar la función de medición A

El diámetro de la esfera de rubí del palpador almacenado en memoria aparece en pantalla

2.- Introducir el diámetro de la esfera del palpador con el que se va a medir. Cuando se ha introducido este dato la pantalla de operaciones muestra:

S 5 - 3 - 2 (número de puntos a introducir requeridos)

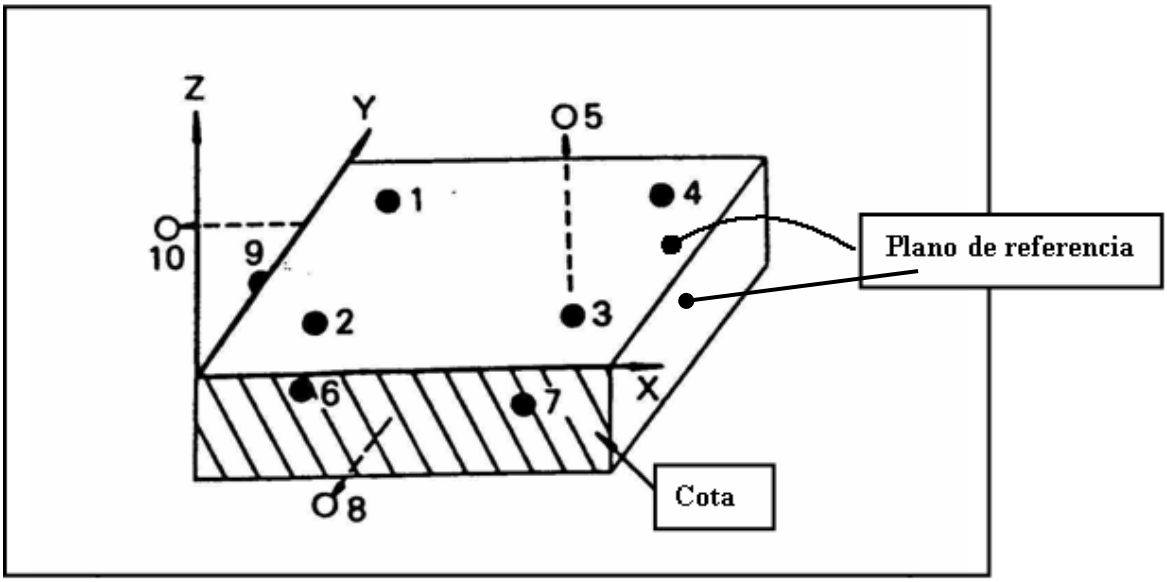
3.- Hacer el alineamiento del plano

4.- Introducir dos puntos a lo largo del lado A y un dummy para establecer una línea de referencia

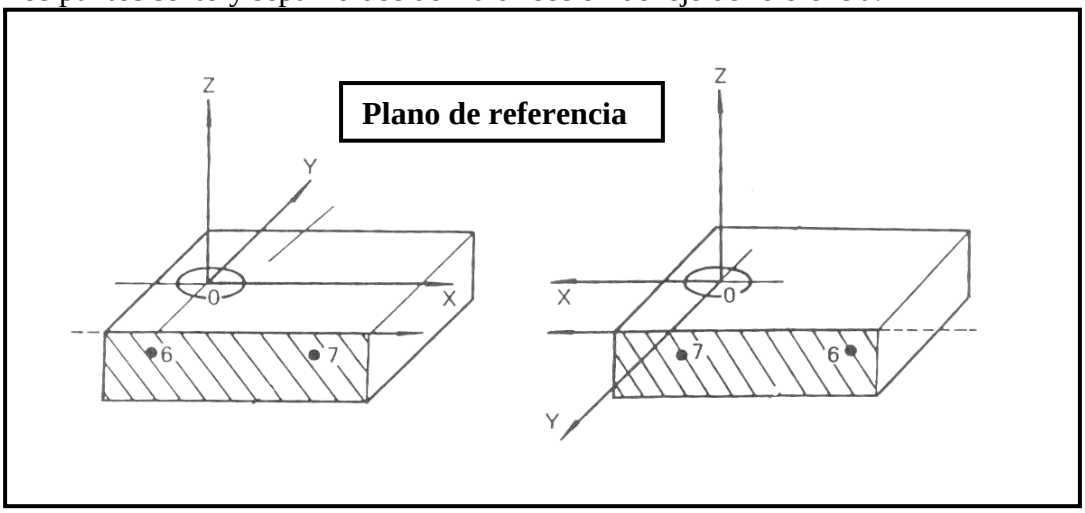
S 0 - 3 - 2 (número de puntos a introducir requeridos)

5.- Introducir un punto a lo largo del eje B y un dummy para determinar la posición del origen

S 0 - 0 - 2



Los puntos sexto y séptimo deciden la dirección del eje de referencia:



6.- Se imprime el siguiente mensaje:

:PATTERN A
PROBE D = 2.998 (p.e.)

MODELO B: Los ejes se sitúan perpendiculares al eje de un cilindro de modo que el eje X pase además por el centro de otro cilindro.

1.- Pulsar la tecla B para seleccionar la función de medición
El diámetro de la esfera de rubí del palpador aparece en pantalla
2.- Introducir el diámetro de la esfera del palpador con el que se va a medir. Cuando se ha introducido este dato la pantalla de operaciones muestra:

S 5 - 5 - 5 (número de puntos a introducir requeridos)

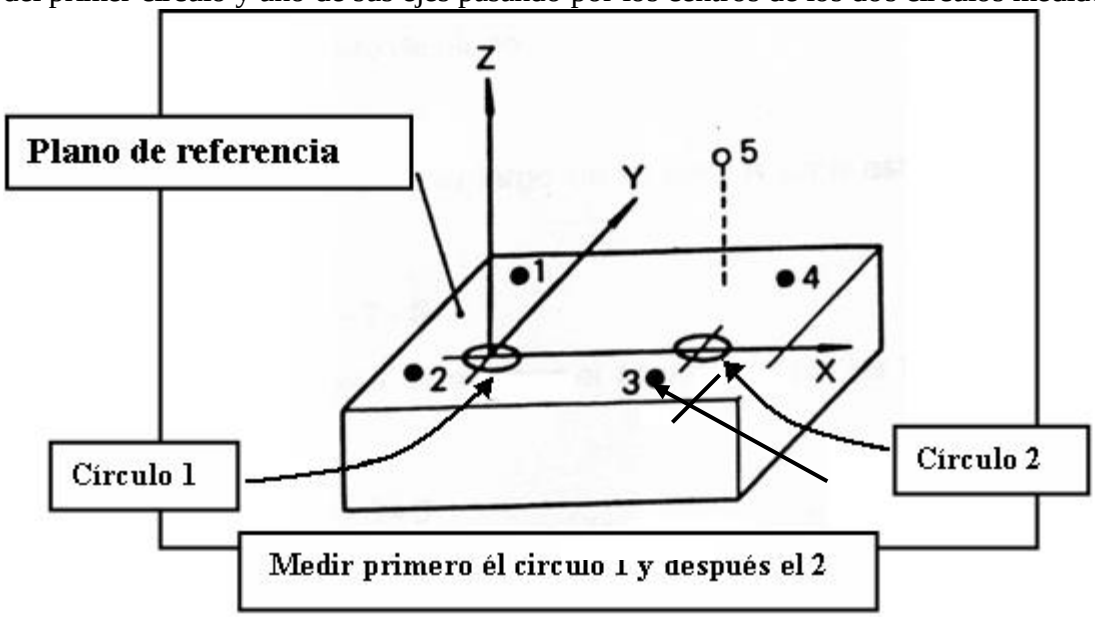
3.- Hacer el alineamiento del plano. Cuando se ha efectuado esta operación aparece en pantalla:

S 0 - 5 - 5

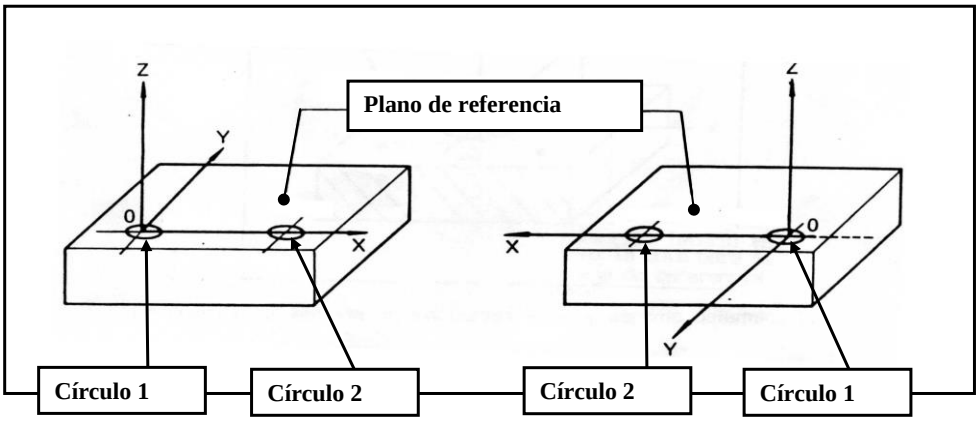
4.- Medir el primer círculo. Cuando se ha efectuado esta operación aparece en pantalla:

S 0 - 0 - 5

5.- Medir el siguiente círculo. El sistema de coordenadas queda establecido con origen el el centro del primer círculo y uno de sus ejes pasando por los centros de los dos círculos medidos.



El orden de medición de los círculos decide la dirección del eje de referencia:



Aparece impreso el siguiente mensaje:

:PATTERN B
PROBE D = 2.988
CIRC D1 = n.nnn D2 = n.nnn

MODELO C: Una cara de la pieza sirve de referencia y el centro de un círculo como origen para establecer el sistema de coordenadas de la pieza

1.- Pulsar la tecla C para seleccionar este modelo de medición
2.- Introducir el diámetro de la probeta
3.- Hacer el alineamiento de plano

S 5 - 2 - 5

4.- Introducir dos puntos a lo largo de la cara A para establecer una línea de referencia

S 0 - 2 - 5

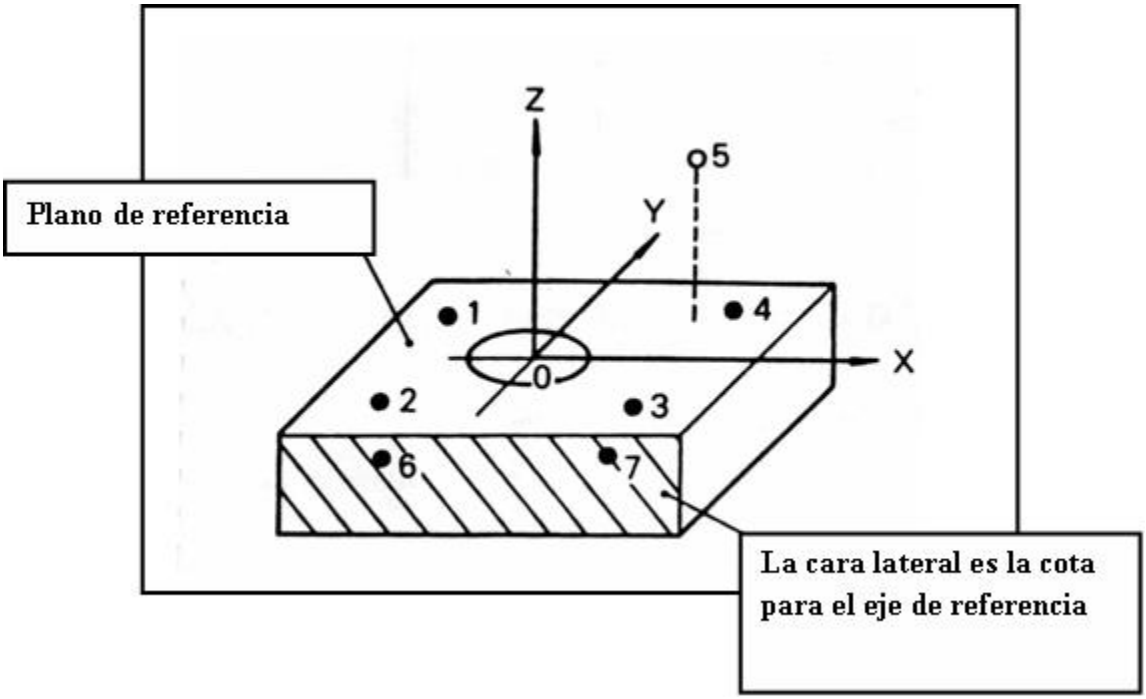
5.- Medir el círculo para determinar el centro, al cual se trasladará el origen

S 0 - 0 - 5

La posición de entrada de los puntos sexto y séptimo determina la dirección de los ejes de referencia.

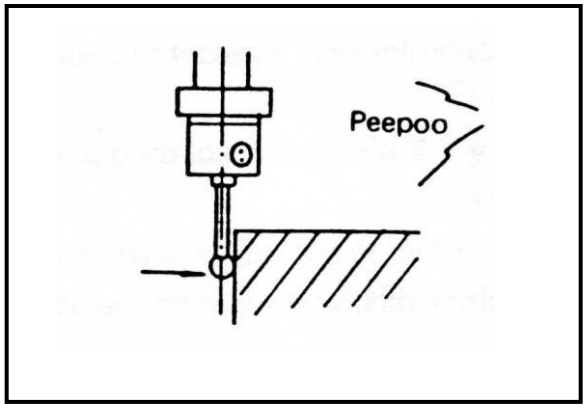
6.- Cuando se establece el sistema de coordenadas, se imprime el siguiente mensaje:

:PATTERN C
PROBE D = 2.998
CIRC D1 = n.nnn

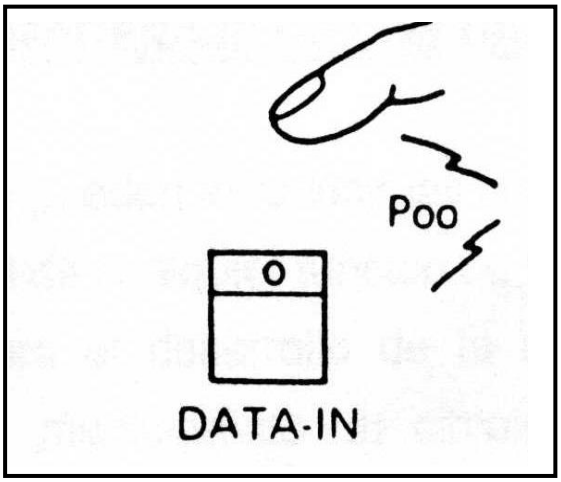


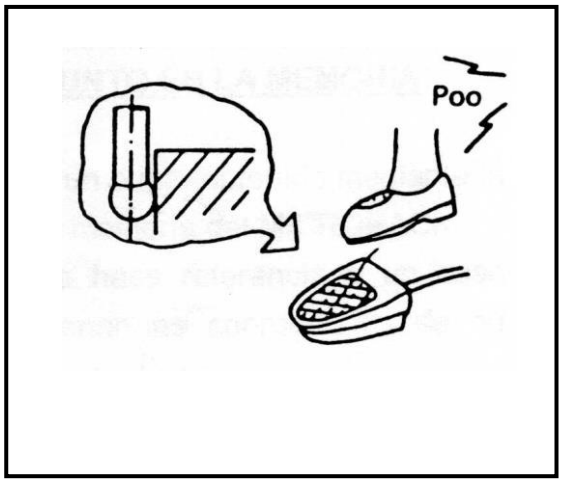
H. ENTRADA DE DATOS DE UN PUNTO

H.1. PALPADO DE UN PUNTO



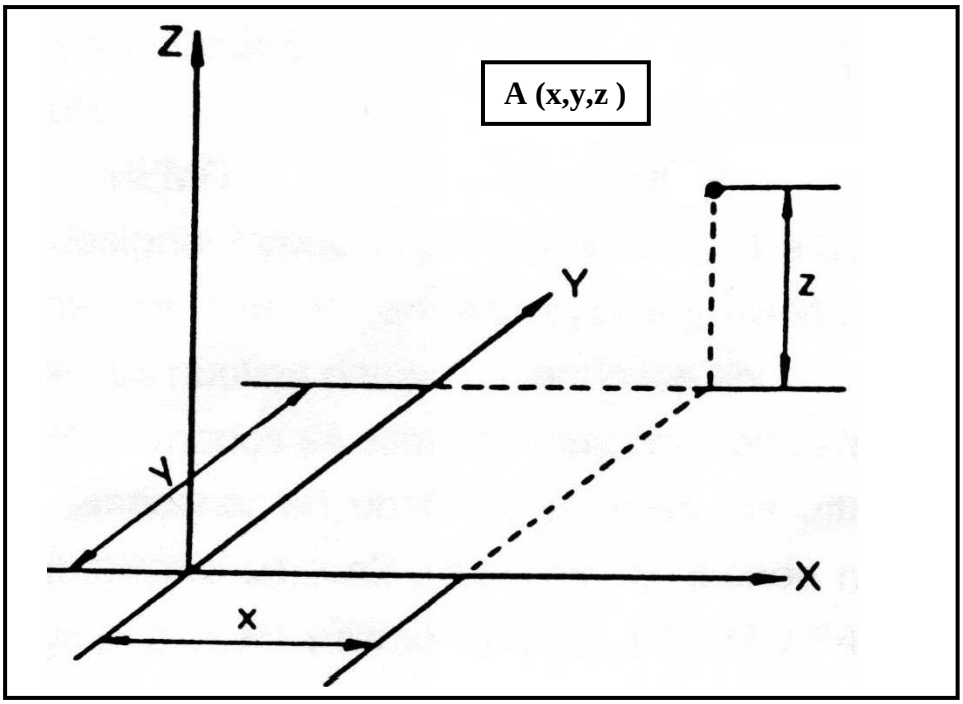
H.2. INTRODUCCIÓN DE UN PUNTO FICTICIO





H.3. INTRODUCCIÓN DE UN PUNTO MEDIANTE TECLADO

Esta función se utiliza para introducir un punto (x,y,z) en términos del sistema de coordenadas de la pieza por medio del teclado.



1.- Pulsar en este orden las teclas FEED y KEY-IN
Aparecerá en pantalla: S n 0.000; siendo n un dígito que representa a cada uno de los ejes de coordenadas: 1= Introducir coordenada X; 2=Introducir coordenada Y; 3=Introducir coordenada Z
2.- Introducir datos por teclado.
Tras introducir los dígitos pulsar la tecla E.
La gama permitida para los valores de X,Y y Z es -999.999 mm < X,Y,Z < 999.999 mm.
Se imprimen los datos introducidos en orden de X,Y y Z.
El punto se considera ya como un punto medido.

H.4. UTILIZACIÓN DE UN PUNTO ALMACENADO EN LA MEMORIA

ALMACENAJE DE LOS DATOS DE UN PUNTO EN LA MEMORIA

Se pueden introducir las coordenadas de un punto obtenido mediante la ejecución de cualquier función de, se hace referencia a un caso concreto: medición de un círculo para almacenar las coordenadas de su centro.

1.- Realizar la medición de un círculo según se ha descrito anteriormente.
2.- Pulsar MEMO. Aparecerá en pantalla:

S 0

3.- Introducir el número de la memoria en que se desee archivar, por ejemplo 1. Finalmente pulsar E. Aparece impreso:

:MEMO.....01, , , ,

Se pueden designar dirección por los números 1 a 30.
Si se designa una dirección en la que haya previamente almacenado un punto se almacenan los nuevos datos, borrando los antiguos.
Los datos almacenados se eliminan desconectando la corriente.
Es posible almacenar en un único proceso los últimos cuatro puntos medidos con la operación: n1 . n2 . n3 . n4 E; siendo n1, n2, n3 y n4 las direcciones para los puntos P1 (último medido), P2, P3 y P4, respectivamente.
En tal caso aparecería impreso:

:MEMO.....n1, n2, n3, n4

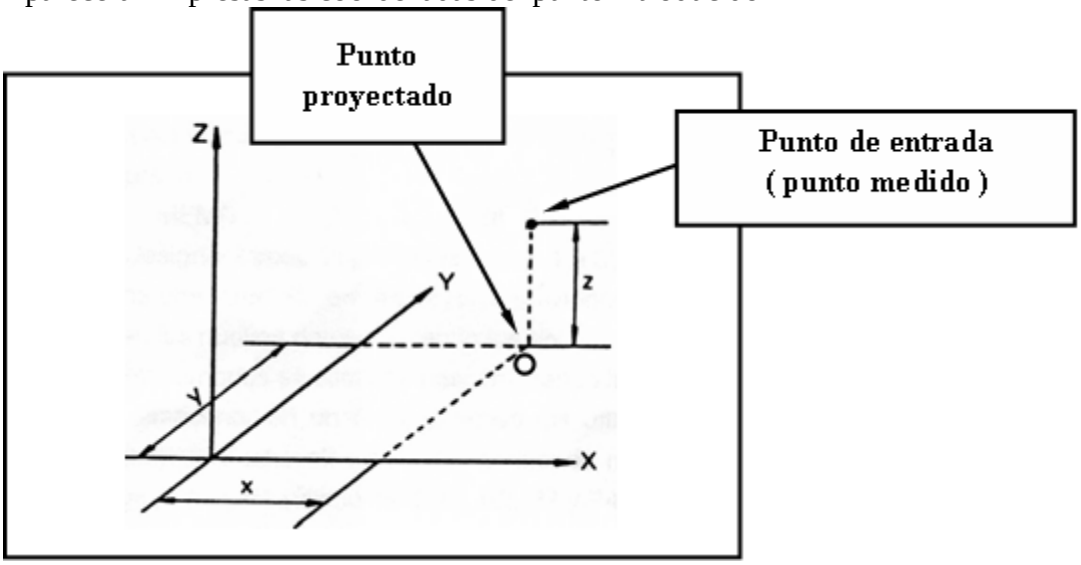
LLAMAMIENTO A LA MEMORIA

Esta función se utiliza para insertar un punto cuyas coordenadas se han almacenado previamente en una posición en la memoria del MICROPAK en un proceso de medición cualquiera.
Para recuperar la función memorizada, se pulsa la tecla RECALL seguida del número de la memoria en que estaba almacenado y la tecla E.

I. OBTENCIÓN DE LAS COORDENADAS DE UN PUNTO PALPADO

Esta función se utiliza para obtener las coordenadas de un punto introducido por palpado. El punto introducido será el centro de la probeta.

1. Coloque la probeta en cualquier posición.
2. Pulsar XYZ
3. Pulsar PT, en pantalla aparecerá el número de puntos necesarios de entrada (1).
4. Introducir el punto que se quiera en el sistema de coordenadas elegido.
Aparecerán impresas las coordenadas del punto introducido



J. MEDICIÓN DE CÍRCULOS Y CILINDROS

Esta función se utiliza para determinar las coordenadas del centro y el diámetro de un agujero circular o de un cilindro, que sea perpendicular al plano de referencia.

1.- Definir un plano de referencia como se indica anteriormente.
2.- Pulsar la tecla CIRC. En pantalla aparecerá:

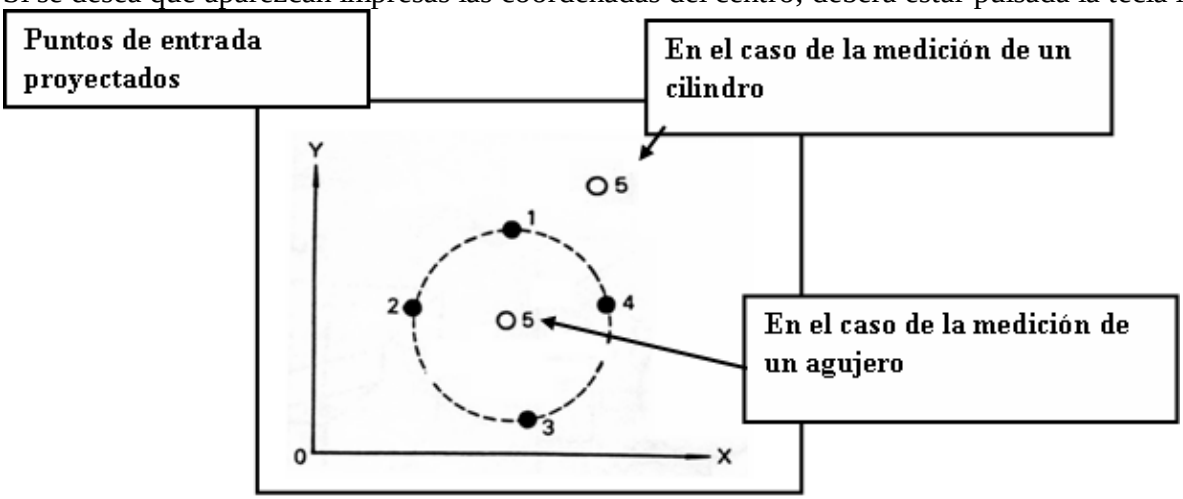
X: ---------
Y: ---------
Z: ---------
S: 5

3.- Introducir los puntos requeridos. Los cuatro primeros, en el perímetro del cilindro o del círculo. El quinto será un dummy. La posición de la entrada ficticia del dummy decide si la medición de la pieza es un agujero o un cilindro. En el primer caso el dummy será interior, y en el segundo, exterior.
Aparece impreso:

:CIRC.....nº operación
D= n.nnn

Siendo: D el diámetro expresado en mm.

Si se desea que aparezcan impresas las coordenadas del centro, deberá estar pulsada la tecla XYZ



K. MEDICIÓN DE LADO

Esta función se usa para determinar una posición de un plano perpendicular a uno de los ejes del sistema de coordenadas de una pieza mediante la introducción de un punto sobre el lado a ser medido y un punto ficticio para establecer la compensación del radio del palpador

1.- Definir un plano de referencia.

Cuando el plano X-Y se designa como plano de referencia mediremos la distancia Z de un plano paralelo al plano de referencia (análogamente mediremos distancias X ó Y cuando se designen como planos de referencia YZ ó ZX).

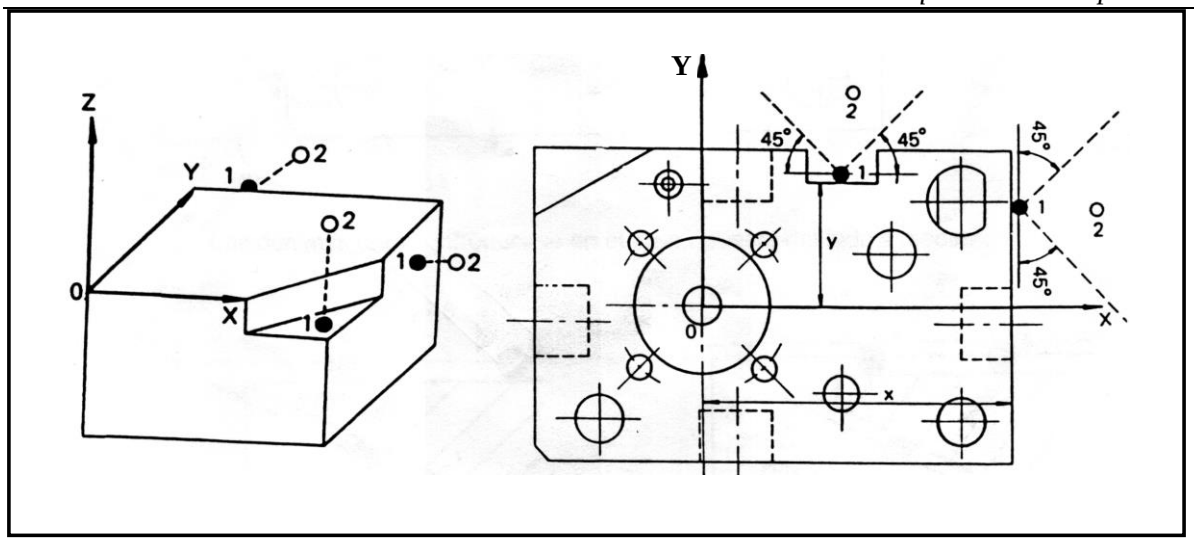
2.- Pulsar la tecla SIDE. En pantalla aparece:

X: -------
Y: -------
Z: -------
S: 2

3.- Introducir los puntos requeridos: uno en el interior y un dummy lo más próximo a la superficie.

Aparece impreso:

:SIDE .....nº operación
Z = n.nnn (ó X = n.nnn ó Y = n.nnn)



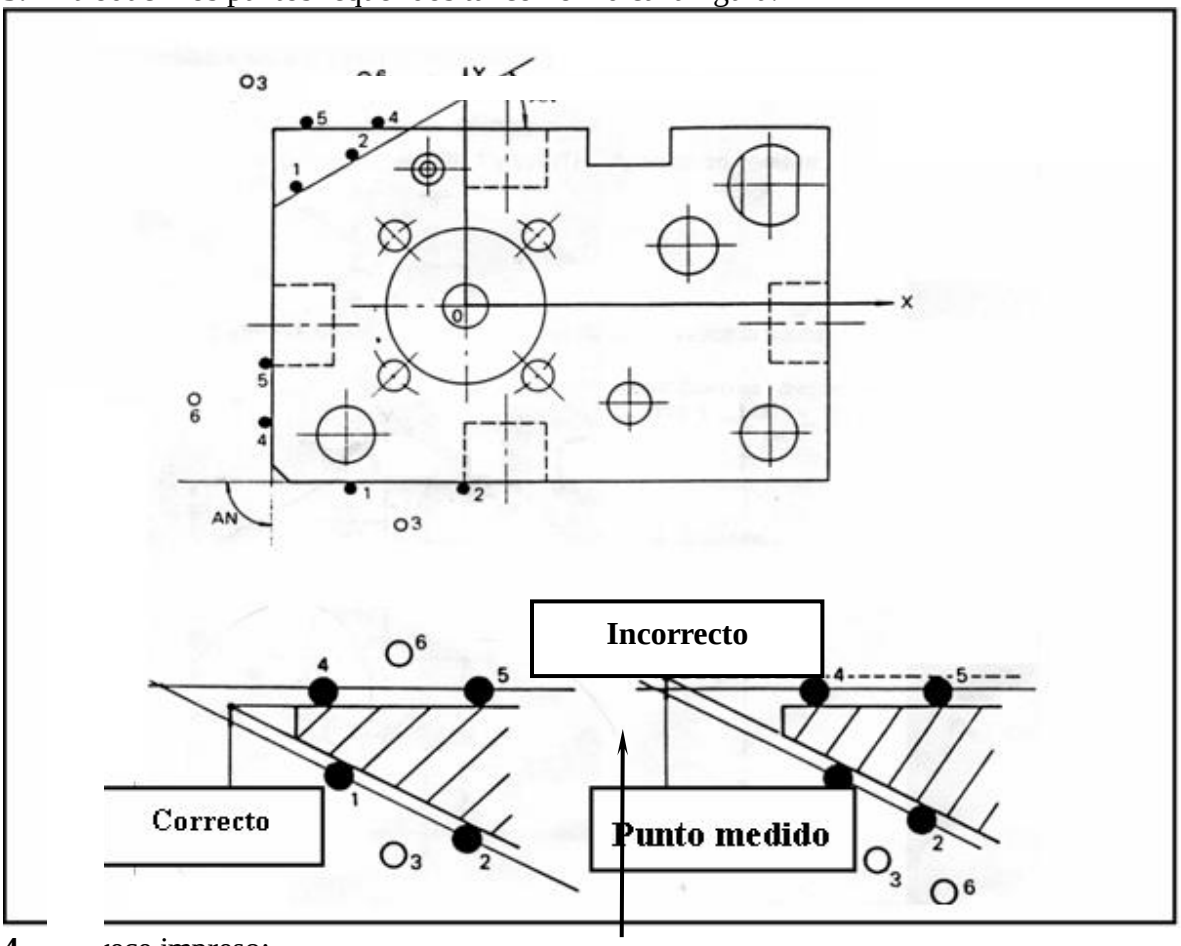
L. MEDICIÓN DE LA INTERSECCIÓN Y DEL ÁNGULO FORMADO

Esta función determina el ángulo (AN) formado por dos planos perpendiculares al de referencia y su punto intersección con él.

1.- Definir un plano de referencia
2.- Pulsar la tecla INTSECT. Aparece en pantalla:

X:	-------
Y:	-------
Z:	-------
S:	6

3.- Introducir los puntos requeridos tal como indica la figura:



4.- Aparece impreso:

:INTSECT.....nº operación
X = n.nnn Y =n.nnn AN = n.nnnn

Siendo AN el ángulo medido en grados, y X e Y las coordenadas del punto intersección de los tres planos, que sólo aparecerán si tenemos encendido el LED de la tecla XYZ
Si la tecla XYZ no está conectada:

:INTSECT.....nº operación
AN = nn.nnnn

M. MEDICIÓN DE ANCHURAS

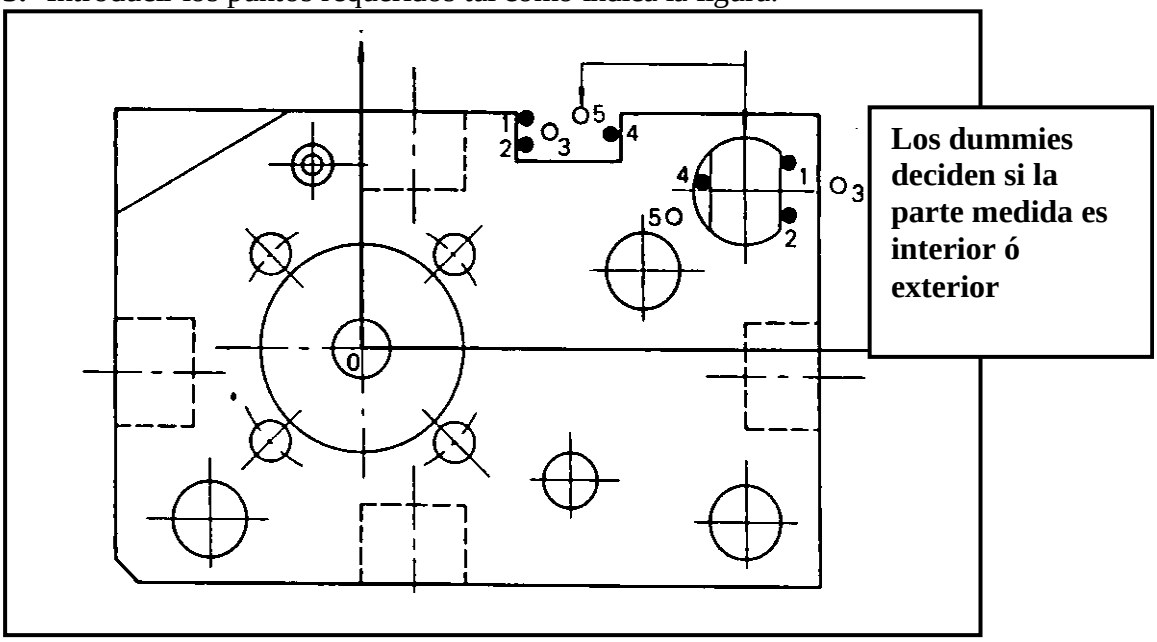
Esta función se utiliza para determinar la distancia entre dos planos paralelos entre sí y perpendiculares al plano de referencia.

1.- Definir un plano de referencia
2.- Pulsar las teclas SHIFT y WIDTH . Aparece en pantalla:

X: ------
Y: ------
Z: ------

S: 5

3.- Introducir los puntos requeridos tal como indica la figura:



4.-Aparece impreso:

:WIDTH .....nº operación

L = n.nnn

N. MEDICIÓN DE BISECTOR

Esta función se emplea para determinar la distancia media entre dos puntos en el espacio (bisector).

1.- Pulsar las teclas SHIFT y WIDTH.

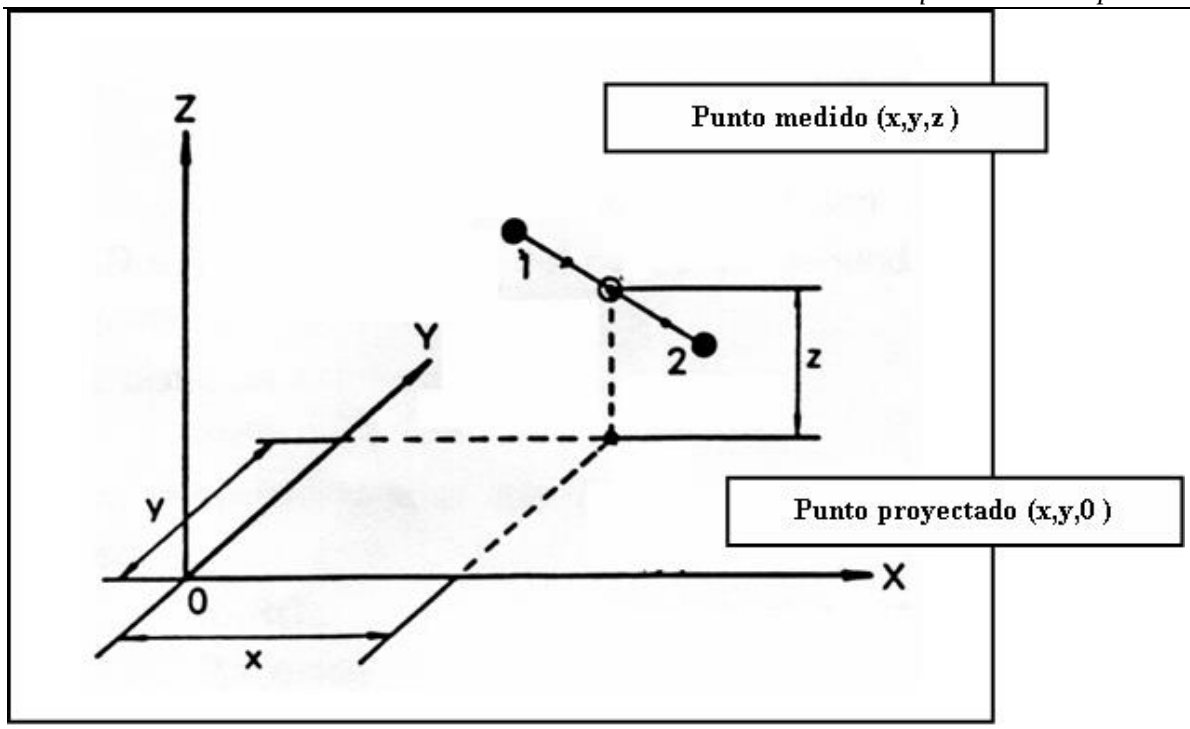
2.- Introducir dos puntos para obtener la proyección del punto medio de esos dos puntos sobre el plano de referencia.

Las coordenadas de los puntos se pueden introducir con el teclado, pulsando KEY-IN., en el orden X, Y, Z. Se pueden introducir como máximo tres dígitos enteros y tres decimales. Después de cada coordenada pulsar E.

3.- Aparece impreso:

:BISECT.....nº operación

X = n.nnn Y = n.nnn



O. MEDIDA DE DISTANCIAS ENTRE CENTROS

Esta función se utiliza para medir la distancia entre ejes de dos cilindros (interiores o exteriores) perpendiculares al plano de referencia definido sobre la pieza.

1.- Definir un plano de referencia
2.- Pulsar la tecla CIRC. Aparece en pantalla:

X:	-------
Y:	-------
Z:	-------
S:	5

3.- Introducir los puntos requeridos como si se fuese a medir un agujero o un cilindro a lo largo del agujero o cilindro que se tome como origen.
Aparece impreso:

:CIRC.....nº operación
D = n.nnn

4.- Pulsar la tecla PITCH
5.- Introducir los 5 puntos requeridos, (4 puntos palpados + 1 dummy) a lo largo del segundo cilindro.
Si estaba seleccionada la opción de impresión de coordenadas polares aparece impreso:

:CIRC.....nº operación

X = n.nnn	Y = n.nnn	D = n.nnn
XD = n.nnn	YD = n.nnn	L = n.nnn

Siendo: XD e YD: coordenadas del centro del segundo círculo respecto del centro del primero

L: distancia entre centros

Si estaba seleccionada la opción de impresión de coordenadas polares aparece impreso:

:CIRC.....nº operación
R = n.nnn A = n.nnn D = n.nnn
XD = n.nnn YD = n.nnn L = n.nnn

Siendo XD e YD: coordenadas del centro del segundo círculo respecto del centro del primero y L: distancia entre centros.

P. GENERACIÓN DEL ARCHIVO DEL PROCESO DE MEDICIÓN

Esta función se usa para almacenar el proceso de medición en un archivo. Esto permitirá utilizarlo posteriormente tantas veces como se quiera, reduciendo el tiempo de medición ya que no será necesario usar operaciones de teclado.

1. Pulsar la tecla LEARN en la columna TEACHING. Aparece impreso LEARNING START
2. Ejecutar el proceso de medición (la programación se efectuará por aprendizaje, es decir, el microprocesador almacenará la secuencia de operaciones introducida)
3. Pulsar END cuando la secuencia de mediciones esté completa. Aparece impreso END. Se observa en pantalla:

S 00 (último número de archivo utilizado)

4. Introducir un número de archivo (nn) bajo el cual se almacenará el proceso. Al finalizar el número pulsar E.
Aparecerá impreso: WRITE FILE NO. nn. Se pueden designar archivos desde el 1 al 60. Si se designa un archivo que ya contiene un proceso almacenado se almacena el nuevo proceso y desaparece el anterior.
Se pueden almacenar en un archivo hasta 50 pasos de procesos.

Q. EJECUCIÓN DEL ARCHIVO DEL PROCESO DE MEDICIÓN

Esta función se usa para ejecutar la medición según un proceso previamente almacenado en un archivo.
1. Pulsar REPEAT en la columna TEACHING. Aparece en pantalla:

S 00 (último número de archivo utilizado)

2.Introducir el número de archivo a ejecutar. Al finalizar el número introducir E.
3.Ejecutar el archivo. Se van iluminando sobre la consola del MICROPAK los LED de la teclas correspondientes a la función de medición que debe ser ejecutada. Durante la ejecución del archivo se observa en la pantalla:

S nn m

Siendo nn el número del proceso de medición y el número de los puntos de entrada requeridos.

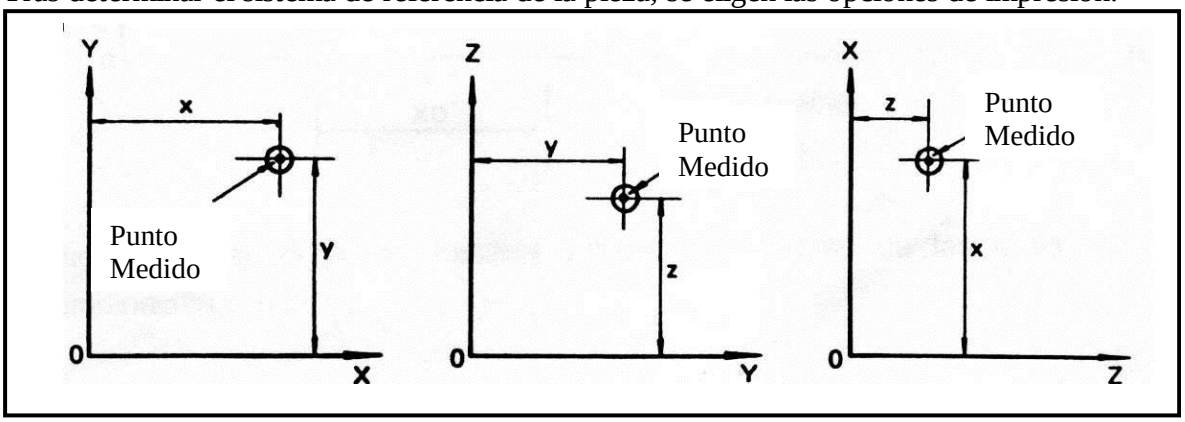
R. INTERRUPCIÓN / REANUDACIÓN DEL PROCESO DE ALMACENAJE / EJECUCIÓN

Esta función se usa para suspender la ejecución de la operación de almacenaje / ejecución de modo que se puedan realizar mediciones independientes del archivo del proceso.

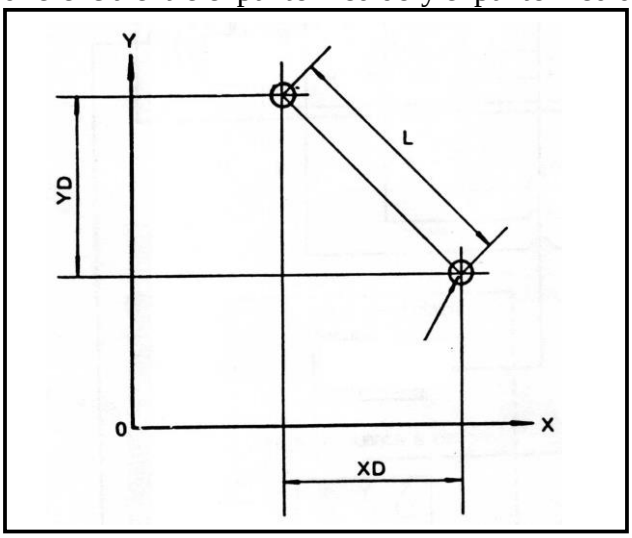
1. En medio de una secuencia de almacenaje / ejecución pulsar INT-RES. Aparece impreso:
<<
2. Ejecutar la medición que no se desea almacenar / que no está almacenada en el archivo en generación / ejecución
3. Pulsar INT-REST cuando ha finalizado la tarea intercalada en el archivo y ajena a éste. Aparece impreso: <<

S. DESIGNACIÓN DE LA IMPRESIÓN

Tras determinar el sistema de referencia de la pieza, se eligen las opciones de impresión:



1. XYZ. Utilizada para imprimir datos en coordenadas cartesianas
2. R-A. Utilizada para imprimir la posición del punto medido en coordenadas polares y el ángulo de diferencia entre el punto medido y el punto medido inmediatamente antes

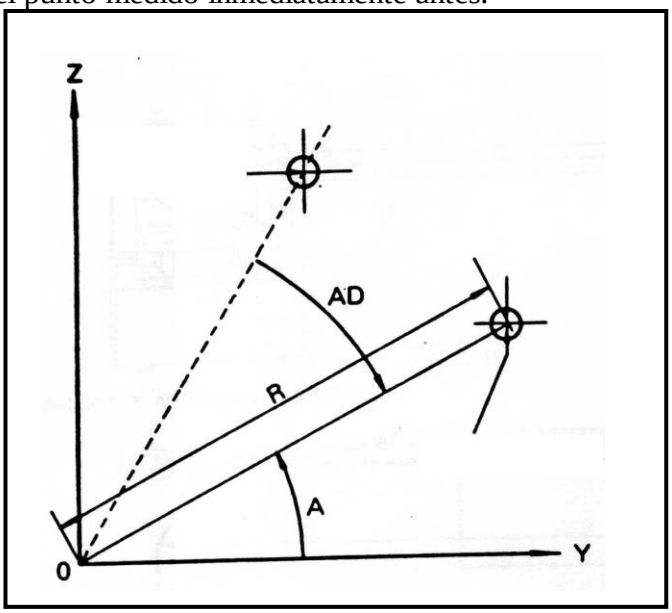


R ==> Distancia entre el origen y el punto medido en el plano de referencia

A ==> Ángulo entre el eje X y la línea formada por el origen y el punto medido

AD ==> Ángulo entre el punto medido y el punto medido inmediatamente antes

3. PITCH. Utilizada para imprimir la distancia y la diferencia de coordenadas entre el punto medido y el punto medido inmediatamente antes.

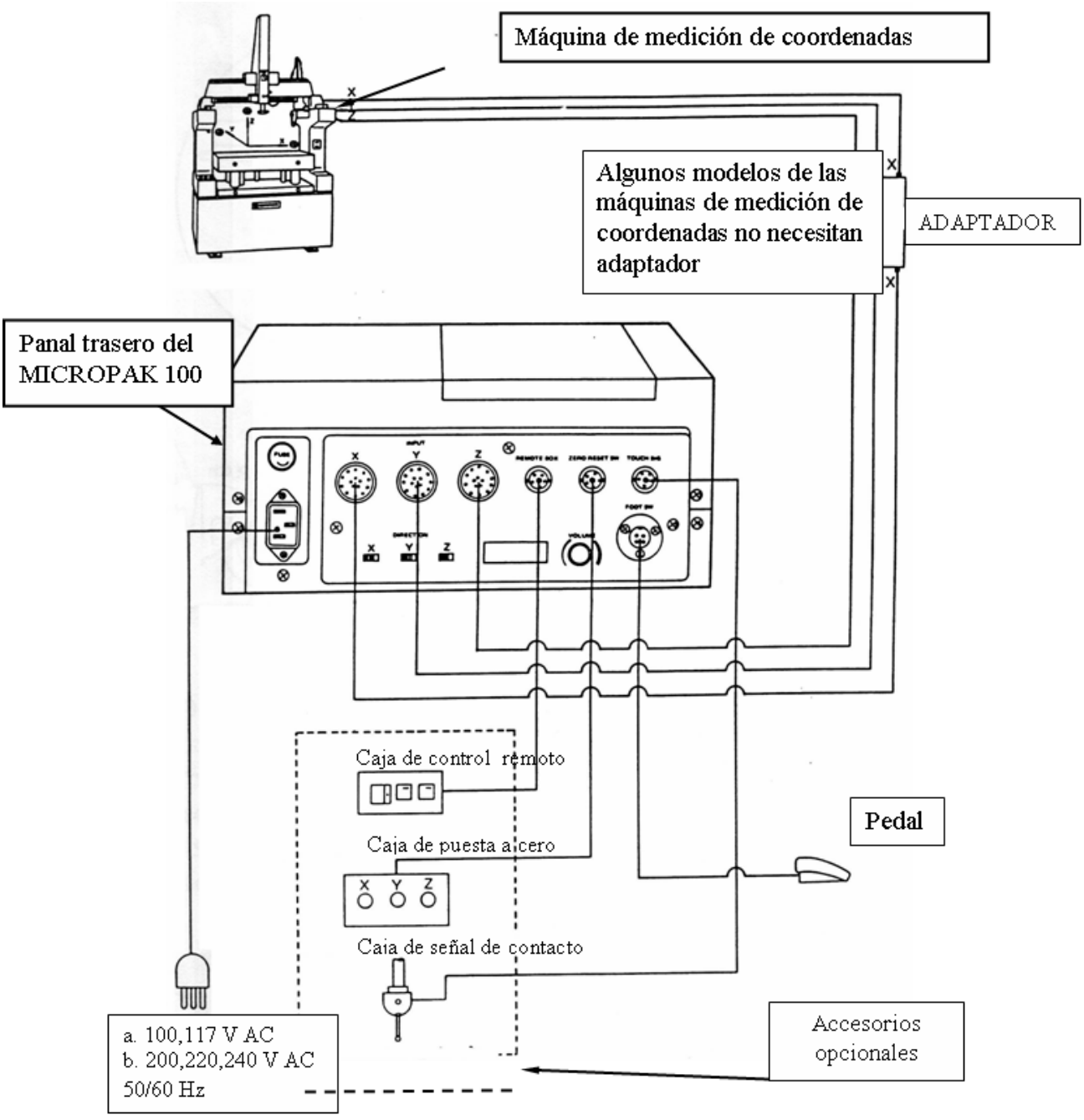


L ==> Distancia entre el punto medido y el punto medido inmediatamente antes

XD ==> Diferencia de coordenadas en el plano de referencia

YD ==> Diferencia de coordenadas en el plano de referencia

APÉNDICE 1: CONEXIONES



Proyecto fin de carrera

Ingeniería Técnica Mecánica

ANEXO III:

CÓDIGO

Autor: José Tovío Ciércoles

Director del proyecto: Mª Rosario González Pedraza

Especialidad: Ingeniería Técnica Mecánica

Fecha: 21 de mayo de 2015

Convocatoria: junio de 2015

```
unit menu;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
  Grids, ExtCtrls, Esfera, Plano, cilindro, cilindro_girado, cono, mates;

type
  { TForm1 }
{Podemos almacenar hasta 100 objetos}
  TObjeto =  (Oesfera, Oplano, Ocilindro, Ocono, Opunto, Orecta,
         Ocilindrogirado, Oconogirado);

  TRegistroObjeto = record
    obj: Tobjeto;
    Param: array[0..6] of real;
  end;

  TListaObjetos = array[0..99] of TRegistroObjeto ;

  TForm1 = class(TForm)
    BCalcular: TButton;
    BAnadir: TButton;
    BCargar1: TButton;
    BGuardar1: TButton;
    BCalcular2: TButton;
    BAnadir2: TButton;
    BCargar2: TButton;
    BGuardar2: TButton;
    BCargar3: TButton;
    BGuardar3: TButton;
    Bdistancias: TButton;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Edit1: TEdit;
```

```
Edit2: TEdit;
Edit3: TEdit;
Edit4: TEdit;
Edit5: TEdit;
GBResultado1: TGroupBox;
GroupBox1: TGroupBox;
GroupBox2: TGroupBox;
GroupBox3: TGroupBox;
GBResultado: TGroupBox;
GBPunto: TGroupBox;
GBDistancias: TGroupBox;
GBIntersecciones: TGroupBox;
Label1: TLabel;
Label2: TLabel;
Label4: TLabel;
Label5: TLabel;
LObjeto1: TLabel;
LParametros: TLabel;
LObjeto: TLabel;
LParametros1: TLabel;
RBIdeal: TRadioButton;
RBRecta: TRadioButton;
RBPunto: TRadioButton;
RBConoGirado: TRadioButton;
RBCilindroGirado: TRadioButton;
RBexterna: TRadioButton;
RBInterna: TRadioButton;
RBDisparo: TRadioButton;
RBesfera: TRadioButton;
RBPlano: TRadioButton;
RBCilindro: TRadioButton;
RBCono: TRadioButton;
RadioGroup1: TRadioGroup;
RadioGroup2: TRadioGroup;
StringGrid1: TStringGrid;
StringGrid2: TStringGrid;
StringGrid3: TStringGrid;
SGListaObjetos: TStringGrid;
procedure BAnadir2Click(Sender: TObject);
procedure BAnadirClick(Sender: TObject);
```

```
    procedure BCargar1Click(Sender: TObject);
    procedure BCargar2Click(Sender: TObject);
    procedure BCargar3Click(Sender: TObject);
    procedure BCilindroClick(Sender: TObject);
    procedure BCilindroGiradoClick(Sender: TObject);
    procedure BConoClick(Sender: TObject);
    procedure BdistanciasClick(Sender: TObject);
    procedure BEsferaClick(Sender: TObject);
    procedure BGuardar1Click(Sender: TObject);
    procedure BGuardar2Click(Sender: TObject);
    procedure BGuardar3Click(Sender: TObject);
    procedure BPlanoClick(Sender: TObject);
    procedure BPuntoClick(Sender: TObject);
    procedure BCalcularClick(Sender: TObject);
    procedure BCalcular2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: char);
    procedure Edit2KeyPress(Sender: TObject; var Key: char);
    procedure Edit3KeyPress(Sender: TObject; var Key: char);
    procedure FormCreate(Sender: TObject);
    procedure LObjeto1Click(Sender: TObject);
    procedure StringGrid1KeyPress(Sender: TObject; var Key: char);
    procedure StringGrid2KeyPress(Sender: TObject; var Key: char);
    procedure StringGrid3KeyPress(Sender: TObject; var Key: char);
  private

  public
    { public declarations }
  end;

var
  Form1: TForm1;
  ListaObjetos: TListaObjetos;
  ObjetoActual,PuntoActual,InterseccionActual: tregistroObjeto;
  nobjetos: integer;

implementation
```

```
{$R *.lfm}

{ TForm1 }

procedure TForm1.BEsferaClick(Sender: TObject);
begin
  FEsfera.show;
end;

procedure TForm1.BGuardar1Click(Sender: TObject);
var
    i: integer;
    guardar: TSaveDialog;
  begin
    guardar:=TSaveDialog.Create(self);
    Guardar.options:=guardar.Options+[ofOverwritePrompt];
    guardar.InitialDir:=GetCurrentDir;
    guardar.Title:='Guardar datos como';
    if guardar.Execute then
    begin
    with TStringList.Create do
    try
      for i:= 0 to StringGrid1.RowCount - 1 do
        begin
        StringGrid1.Rows[i].Delimiter:=';';
        Add(StringGrid1.Rows[i].DelimitedText);
        end;
        SaveToFile(guardar.Filename);
    finally
      Free;
    end;
    ShowMessage('Datos guardados en: ' + guardar.Filename);
    end;
    guardar.Free;
  end;

procedure TForm1.BGuardar2Click(Sender: TObject);
var
    i: integer;
```

```
    guardar: TSaveDialog;
  begin
    guardar:=TSaveDialog.Create(self);
    Guardar.options:=guardar.Options+[ofOverwritePrompt];
    guardar.InitialDir:=GetCurrentDir;
    guardar.Title:='Guardar datos como';
    if guardar.Execute then
    begin
    with TStringList.Create do
    try
      for i:= 0 to StringGrid2.RowCount - 1 do
        begin
        StringGrid2.Rows[i].Delimiter:=';';
        Add(StringGrid2.Rows[i].DelimitedText);
        end;
        SaveToFile(guardar.Filename);
    finally
     Free;
    end;
    ShowMessage('Datos guardados en: ' + guardar.Filename);
    end;
    guardar.Free;
  end;

procedure TForm1.BGuardar3Click(Sender: TObject);
var
    i: integer;
    guardar: TSaveDialog;
  begin
    guardar:=TSaveDialog.Create(self);
    Guardar.options:=guardar.Options+[ofOverwritePrompt];
    guardar.InitialDir:=GetCurrentDir;
    guardar.Title:='Guardar objetos como';
    if guardar.Execute then
    begin
    with TStringList.Create do
    try
      for i:= 0 to SGListaObjetos.RowCount - 1 do
        begin
        SGListaObjetos.Rows[i].Delimiter:=';';
```

```
      Add(SGListaObjetos.Rows[i].DelimitedText);
      end;
      SaveToFile(guardar.Filename);
   finally
    Free;
   end;
   ShowMessage('Objetos guardados en: ' + guardar.Filename);
   end;
   guardar.Free;
 end;



procedure TForm1.BCilindroClick(Sender: TObject);
begin
  FCilindro.show;
end;

procedure TForm1.BCargar1Click(Sender: TObject);
var
  i,j: integer;
  abrir: TOpenDialog;
begin
  abrir:=TOpenDialog.Create(self);
  abrir.InitialDir:=GetCurrentDir;
  abrir.Title:='Abrir archivo de datos';
  if abrir.Execute then
  begin
  with TStringList.Create do
  try
    LoadFromFile(abrir.Filename);
    StringGrid1.RowCount:= Count;
    {Borrar primero el contenido}
    for i:=1 to StringGrid1.RowCount-1 do
    for j:=0 to StringGrid1.ColCount-1 do
      StringGrid1.Cells[j,i]:='';
    {Cargar los datos}
    for i:= 0 to Count - 1 do
      begin
```

```
      StringGrid1.Rows[i].Delimiter:=';';
      StringGrid1.Rows[i].DelimitedText:= Strings[i];
      end;
  finally
    Free;
  end;
  ShowMessage('Datos cargados de: ' + abrir.Filename);
  end;
  abrir.Free;
end;

procedure TForm1.BCargar2Click(Sender: TObject);
var
  i: integer;
  abrir: TOpenDialog;
begin
  abrir:=TOpenDialog.Create(self);
  abrir.InitialDir:=GetCurrentDir;
  abrir.Title:='Abrir archivo de datos';
  if abrir.Execute then
  begin
  with TStringList.Create do
  try
    LoadFromFile(abrir.Filename);
    StringGrid2.RowCount:= Count;
    for i:= 0 to Count - 1 do
      begin
      StringGrid2.Rows[i].Delimiter:=';';
      StringGrid2.Rows[i].DelimitedText:= Strings[i];
      end;
  finally
    Free;
  end;
  ShowMessage('Datos cargados de: ' + abrir.Filename);
  end;
  abrir.Free;
end;

procedure TForm1.BCargar3Click(Sender: TObject);
var
```

```
  i,j: integer;
  nobjetoscargados: integer;
  abrir: TOpenDialog;
  cadena: String;
begin
  abrir:=TOpenDialog.Create(self);
  abrir.InitialDir:=GetCurrentDir;
  abrir.Title:='Abrir archivo de datos';
  nobjetoscargados:=0;
  if abrir.Execute then
  begin
  with TStringList.Create do
  try
    LoadFromFile(abrir.Filename);
    SGListaObjetos.RowCount:= Count;

    for i:= 0 to Count - 1 do
      begin
      SGListaObjetos.Rows[i].Delimiter:=';';
      SGListaObjetos.Rows[i].DelimitedText:= Strings[i];
      end;
    for i:=0 to count - 1 do
      if not (SGListaObjetos.Cells[0,i] = '') then nobjetoscargados := nobjetoscargados + 1;
    nobjetoscargados:=nobjetoscargados-2;

    {Almacenarlos en la lista de objetos}
    for i:=0 to nobjetoscargados do
      begin
        cadena := SGListaObjetos.Cells[1,i+1];
        case cadena of
          'Esfera': ListaObjetos[i].Obj:= Oesfera;
          'Cono': ListaObjetos[i].Obj:= OCono;
          'Plano': ListaObjetos[i].Obj:= OPlano;
          'Cilindro': ListaObjetos[i].Obj:= OCilindro;
          'Cono Gir.': ListaObjetos[i].Obj:= OconoGirado;
          'Cil. Gir.': ListaObjetos[i].Obj:= OcilindroGirado;
          'Punto': ListaObjetos[i].Obj:= Opunto;
        end;
        for j:=0 to 5 do
          if not(SGListaObjetos.Cells[j+2,i+1] = '')
```

```
            then ListaObjetos[i].Param[j]:= StrToFloat(SGListaObjetos.Cells[j+2,i+1]);
        nobjetos:=nobjetoscargados+1;
      end;
  finally
    Free;
  end;
  ShowMessage('Objetos cargados de: ' + abrir.Filename);

 { ShowMessage('Segundo objeto ' + FloatToStr(ListaObjetos[1].Param[0]) + #10+#13+
  FloatToStr(ListaObjetos[1].Param[1]) + #10+#13+FloatToStr(ListaObjetos[1].Param[2]) + #10+#13);  }

  end;
  abrir.Free;
end;

procedure TForm1.BAnadirClick(Sender: TObject);
var
  cadena: string;
begin
  ListaObjetos[nobjetos]:=ObjetoActual;
  nobjetos:=nobjetos+1;
  SGListaObjetos.Cells[0,nobjetos]:= IntToStr(nobjetos-1);
  case ObjetoActual.obj of
    Oesfera: cadena:='Esfera';
    OPlano: cadena:='Plano';
    OCilindro: cadena:='Cilindro';
    OCono: cadena:='Cono';
    OConoGirado: cadena:='Cono Gir.';
    OCilindroGirado: cadena:='Cil. Gir.';
    OPunto: cadena:='Punto';
    ORecta: cadena:='Recta';
  end;
  SGListaObjetos.Cells[1,nobjetos]:= cadena;
  SGListaObjetos.Cells[2,nobjetos]:= FloatToStr(ObjetoActual.Param[0]);
  SGListaObjetos.Cells[3,nobjetos]:= FloatToStr(ObjetoActual.Param[1]);
  SGListaObjetos.Cells[4,nobjetos]:= FloatToStr(ObjetoActual.Param[2]);

  if (ObjetoActual.obj =Oplano) then begin
  SGListaObjetos.Cells[5,nobjetos]:= FloatToStr(ObjetoActual.Param[3]);
  SGListaObjetos.Cells[6,nobjetos]:= FloatToStr(ObjetoActual.Param[4]);
```

```
  SGListaObjetos.Cells[7,nobjetos]:= FloatToStr(ObjetoActual.Param[5]);

  end;

  if (ObjetoActual.obj =ORecta) then begin
  SGListaObjetos.Cells[5,nobjetos]:= FloatToStr(ObjetoActual.Param[3]);
  SGListaObjetos.Cells[6,nobjetos]:= FloatToStr(ObjetoActual.Param[4]);
  SGListaObjetos.Cells[7,nobjetos]:= FloatToStr(ObjetoActual.Param[5]);

  end;

  if (ObjetoActual.obj =Oesfera) then begin
  SGListaObjetos.Cells[5,nobjetos]:= FloatToStr(ObjetoActual.Param[3]);
  end;

  if (ObjetoActual.obj =Ocono) then begin
  SGListaObjetos.Cells[5,nobjetos]:= FloatToStr(ObjetoActual.Param[3]);
  end;

  if (ObjetoActual.obj =OCilindroGirado) then begin
  SGListaObjetos.Cells[5,nobjetos]:= FloatToStr(ObjetoActual.Param[3]);
  SGListaObjetos.Cells[6,nobjetos]:= FloatToStr(ObjetoActual.Param[4]);
  SGListaObjetos.Cells[7,nobjetos]:= FloatToStr(ObjetoActual.Param[5]);
  SGListaObjetos.Cells[8,nobjetos]:= FloatToStr(ObjetoActual.Param[6]);
  end;

  if (ObjetoActual.obj =OConoGirado) then begin
  SGListaObjetos.Cells[5,nobjetos]:= FloatToStr(ObjetoActual.Param[3]);
  SGListaObjetos.Cells[6,nobjetos]:= FloatToStr(ObjetoActual.Param[4]);
  SGListaObjetos.Cells[7,nobjetos]:= FloatToStr(ObjetoActual.Param[5]);
  SGListaObjetos.Cells[8,nobjetos]:= FloatToStr(ObjetoActual.Param[6]);
  end;


end;

procedure TForm1.BAnadir2Click(Sender: TObject);
var
  cadena: string;
begin
```

```
  ListaObjetos[nobjetos]:=PuntoActual;
  nobjetos:=nobjetos+1;
  SGListaObjetos.Cells[0,nobjetos]:= IntToStr(nobjetos-1);
  SGListaObjetos.Cells[1,nobjetos]:= 'Punto';
  SGListaObjetos.Cells[2,nobjetos]:= FloatToStr(PuntoActual.Param[0]);
  SGListaObjetos.Cells[3,nobjetos]:= FloatToStr(PuntoActual.Param[1]);
  SGListaObjetos.Cells[4,nobjetos]:= FloatToStr(PuntoActual.Param[2]);
end;


procedure TForm1.BCilindroGiradoClick(Sender: TObject);
begin
  FCilindroGirado.show;
end;

procedure TForm1.BConoClick(Sender: TObject);
begin
  FCono.show;
end;

procedure TForm1.BdistanciasClick(Sender: TObject);
var
  objeto1,objeto2,i: integer;
  distancia, distancia2: real;
  v1,v2,w1,w2,punto1,punto2:tvector3;
  calculado: boolean;
  cadena: String;
begin
  calculado:=false;
  objeto1:=StrToInt(Edit2.Text);
  objeto2:=StrToInt(Edit3.Text);
  if (objeto1 >= 0) and (objeto1<nobjetos) and (objeto2 >= 0) and (objeto2<nobjetos) and (not (objeto1=objeto2))
  then
  begin
    {Almacenamos un punto de cada objeto}
  for i:= 0 to 2 do
    begin
      punto1[i]:=ListaObjetos[Objeto1].Param[i];
      punto2[i]:=ListaObjetos[Objeto2].Param[i];
    end;
```

```
{En un cilindro recto el punto tiene coordenada z =0}
  if (ListaObjetos[objeto1].obj = OCilindro)  then punto1[2]:=0;
  if (ListaObjetos[objeto2].obj = OCilindro)  then punto2[2]:=0;

   {almacenamos un vecto de cada objeto que sea necesario}
    {Para el cilindro y el cono girados}
  if ((ListaObjetos[objeto1].obj = OCilindroGirado) or (ListaObjetos[objeto1].obj = OConoGirado))
     then
     for i:= 0 to 2 do
        v1[i]:=ListaObjetos[Objeto1].Param[i+4];

  if ((ListaObjetos[objeto2].obj = OCilindroGirado) or (ListaObjetos[objeto2].obj = OConoGirado))
     then
     for i:= 0 to 2 do
        v2[i]:=ListaObjetos[Objeto2].Param[i+4];
   {Para el cilindro y el cono rectos, la coordenada z es uno y las otras cero}
 if ((ListaObjetos[objeto1].obj = OCilindro) or (ListaObjetos[objeto1].obj = OCono))
     then    begin
     for i:= 0 to 2 do
        v1[i]:=0;
     v1[2]:=1;
     end;

  if ((ListaObjetos[objeto2].obj = OCilindro) or (ListaObjetos[objeto2].obj = OCono))
     then  begin
     for i:= 0 to 2 do
        v2[i]:=0;
     v2[2]:=1;
     end;


   {Para la recta y el plano}
  if ((ListaObjetos[objeto1].obj = OPlano) or (ListaObjetos[objeto1].obj = ORecta))
     then
     for i:= 0 to 2 do
        v1[i]:=ListaObjetos[Objeto1].Param[i+3];



  if ((ListaObjetos[objeto2].obj = OPlano) or (ListaObjetos[objeto2].obj = ORecta))
```

```
      then
          for i:= 0 to 2 do
             v2[i]:=ListaObjetos[Objeto2].Param[i+3];

    if ((((ListaObjetos[objeto1].obj = OPunto) or (ListaObjetos[objeto1].obj = OEsfera) or (ListaObjetos[objeto1].obj =
OCono) or (ListaObjetos[objeto1].obj = OConoGirado)) and ((ListaObjetos[objeto2].obj = OEsfera) or
(ListaObjetos[objeto2].obj = OConoGirado) or (ListaObjetos[objeto2].obj = OCono) or (ListaObjetos[objeto2].obj =
OPunto)))
      or (((ListaObjetos[objeto2].obj = OPunto) or (ListaObjetos[objeto2].obj = OEsfera) or (ListaObjetos[objeto2].obj =
OCono) or (ListaObjetos[objeto2].obj = OConoGirado))and ((ListaObjetos[objeto1].obj = OEsfera) or
(ListaObjetos[objeto1].obj = OConoGirado) or (ListaObjetos[objeto1].obj = OCono) or (ListaObjetos[objeto1].obj =
OPunto)))
    )  then
    begin
    cadena:='Distancia entre centros, puntos o vértices: ' + FloatToStr(distanciaPuntoPunto(punto1,punto2));
    calculado:=True;
    end;


    if ((((ListaObjetos[objeto1].obj = OPunto) or (ListaObjetos[objeto1].obj = OEsfera) or (ListaObjetos[objeto1].obj =
OCono) or (ListaObjetos[objeto1].obj = OConoGirado)) and ((ListaObjetos[objeto2].obj = OCono) or
(ListaObjetos[objeto2].obj = OConoGirado) or (ListaObjetos[objeto2].obj = OCilindro) or (ListaObjetos[objeto2].obj =
OCilindroGirado) or (ListaObjetos[objeto2].obj = ORecta))))  then
    begin
    {cadena:=cadena  + #10+#13+ FloatToStr(punto1[0])
                + #10+#13+ FloatToStr(punto1[1])
                + #10+#13+ FloatToStr(punto1[2])
                + #10+#13+ FloatToStr(punto2[0])
                + #10+#13+ FloatToStr(punto2[1])
                + #10+#13+ FloatToStr(punto2[2])
                + #10+#13+ FloatToStr(v2[0])
                + #10+#13+ FloatToStr(v2[1])
                + #10+#13+ FloatToStr(v2[2]);  }
            cadena:=cadena  +  #10+#13+'Distancia  entre  centro  del  objeto  1  y  eje  del  objeto  2:  '  +
FloatToStr(distanciaPuntoRecta(punto1,punto2,v2));
   { cadena:=cadena  + #10+#13+ FloatToStr(punto1[0])
                + #10+#13+ FloatToStr(punto1[1])
                + #10+#13+ FloatToStr(punto1[2])
                + #10+#13+ FloatToStr(punto2[0])
                + #10+#13+ FloatToStr(punto2[1])
```

```
            + #10+#13+ FloatToStr(punto2[2])
            + #10+#13+ FloatToStr(v2[0])
            + #10+#13+ FloatToStr(v2[1])
            + #10+#13+ FloatToStr(v2[2]);  }
    calculado:=True;
    end;

     if ((((ListaObjetos[objeto2].obj = OPunto) or (ListaObjetos[objeto2].obj = OEsfera) or (ListaObjetos[objeto2].obj =
OCono) or (ListaObjetos[objeto2].obj = OConoGirado)) and ((ListaObjetos[objeto1].obj = OCono) or
(ListaObjetos[objeto1].obj = OConoGirado) or (ListaObjetos[objeto1].obj = OCilindro) or (ListaObjetos[objeto1].obj =
OCilindroGirado) or (ListaObjetos[objeto1].obj = ORecta))))   then
    begin
      {  cadena:=cadena   + #10+#13+ FloatToStr(punto1[0])
            + #10+#13+ FloatToStr(punto1[1])
            + #10+#13+ FloatToStr(punto1[2])
            + #10+#13+ FloatToStr(punto2[0])
            + #10+#13+ FloatToStr(punto2[1])
            + #10+#13+ FloatToStr(punto2[2])
            + #10+#13+ FloatToStr(v2[0])
            + #10+#13+ FloatToStr(v2[1])
            + #10+#13+ FloatToStr(v2[2]);    }
          cadena:=cadena  +  #10+#13+'Distancia  entre  centro  del  objeto  2  y  eje  del  objeto  1:  '  +
FloatToStr(distanciaPuntoRecta(punto2,punto1,v1));
      {  cadena:=cadena   + #10+#13+ FloatToStr(punto1[0])
            + #10+#13+ FloatToStr(punto1[1])
            + #10+#13+ FloatToStr(punto1[2])
            + #10+#13+ FloatToStr(punto2[0])
            + #10+#13+ FloatToStr(punto2[1])
            + #10+#13+ FloatToStr(punto2[2])
            + #10+#13+ FloatToStr(v2[0])
            + #10+#13+ FloatToStr(v2[1])
            + #10+#13+ FloatToStr(v2[2]); }
    calculado:=True;
    end;

     if ((((ListaObjetos[objeto1].obj = OPunto) or (ListaObjetos[objeto1].obj = OEsfera) or (ListaObjetos[objeto1].obj =
OCono) or (ListaObjetos[objeto1].obj = OConoGirado)) and ((ListaObjetos[objeto2].obj = OPlano) )))   then
    begin
            cadena:=cadena  +  #10+#13+'Distancia  entre  centro  del  objeto  1  y  el  plano  2:  '  +
FloatToStr(distanciaPuntoplano(punto1,punto2,v2));
```

```
    calculado:=True;
    end;

     if ((((ListaObjetos[objeto2].obj = OPunto) or (ListaObjetos[objeto2].obj = OEsfera) or (ListaObjetos[objeto2].obj =
OCono) or (ListaObjetos[objeto2].obj = OConoGirado)) and ((ListaObjetos[objeto1].obj = OPlano) )))   then
    begin
                cadena:=cadena + #10+#13+'Distancia entre centro del objeto 2 y el plano 1: ' +
FloatToStr(distanciaPuntoplano(punto2,punto1,v1));
    calculado:=True;
    end;

                if ((((ListaObjetos[objeto1].obj = OCono) or (ListaObjetos[objeto1].obj = OConoGirado) or
(ListaObjetos[objeto1].obj = OCilindro) or (ListaObjetos[objeto1].obj = OCilindroGirado) or
(ListaObjetos[objeto1].obj = ORecta)))
                and ((ListaObjetos[objeto2].obj = OCono) or (ListaObjetos[objeto2].obj = OConoGirado) or
(ListaObjetos[objeto2].obj = OCilindro) or (ListaObjetos[objeto2].obj = OCilindroGirado) or
(ListaObjetos[objeto2].obj = ORecta)))   then
    begin
    cadena:=cadena + #10+#13+'Distancia entre ejes: ' + FloatToStr(distanciaRectaRecta(punto1,punto2,v1,v2));
    calculado:=True;
    end;




  end;

  if calculado = true then showmessage(Cadena)
    else  showmessage('No puede calcularse la distancia. Solo puede calcularse'
    +' la distancia entre puntos, punto plano, punto recta');

end;

procedure TForm1.BPlanoClick(Sender: TObject);
begin
  FPlano.show;
end;

procedure TForm1.BPuntoClick(Sender: TObject);
begin
```

```
end;

procedure TForm1.BCalcularClick(Sender: TObject);
var
A: tmatriz4x3;
A1,matA,Ainv:tmatriz3x3;

B, B1, B2: tmatriz20x3;
v,punto: tvector3;
i,j, npuntos, iteraciones, ncolumnas, ncolumna2, npuntos2: integer;
radio, radio1, d, error: real;
sol: tvector4;
sol1,sol1bis,puntoq, puntoqbis, puntop, puntoqgirado, centro, u1, u2,u3,c,bi,bibis: tvector3;
v1 : tvector2;
minz,distanciACono, dist: real;
posminz:integer;
errorenmatriz:boolean;


begin
{Contamos el numero de filas no vacias del StringGrid}
  ncolumnas:=0;
  for i:=0 to 19 do
    begin
      if StringGrid1.Cells[0,i] <> '' then ncolumnas:=ncolumnas+1;
    end;
  ncolumnas:=ncolumnas-1;

  npuntos:=ncolumnas;
  {Iniciamos la matriz 20x3 con los elementos del StringGrid y ceros el resto}
for i:=0 to 19 do
  for j:=0 to 2 do
    B[i,j]:= 0;


{Contamos el numero de filas no vacias del segundo StringGrid}
  ncolumna2:=0;
  for i:=0 to 19 do
    begin
```

```
    if StringGrid2.Cells[0,i] <> '' then ncolumna2:=ncolumna2+1;
   end;
  ncolumna2:=ncolumna2-1;

  npuntos2:=ncolumna2;
  {Iniciamos la matriz 20x3 con los elementos del segundo StringGrid y ceros el resto}
for i:=0 to 19 do
  for j:=0 to 2 do
    B2[i,j]:= 0;

{Iniciamos ya ambas matrices}
try
for i:=0 to npuntos-1 do
  for j:=0 to 2 do
    B[i,j]:= StrToFloat(StringGrid1.Cells[j,i+1]);

for i:=0 to npuntos2-1 do
  for j:=0 to 2 do
    B2[i,j]:= StrToFloat(StringGrid2.Cells[j,i+1]);

  if RBPunto.Checked=True then
   begin
      PuntoQ[0]:=StrToFloat(StringGrid1.Cells[0,1]);
      PuntoQ[1]:=StrToFloat(StringGrid1.Cells[1,1]);
      PuntoQ[2]:=StrToFloat(StringGrid1.Cells[2,1]);
      for i:= 0 to 2 do
        ObjetoActual.Param[i]:= PuntoQ[i];
      ObjetoActual.obj:=OPunto;
      LObjeto.Caption:= 'Objeto: Punto';
      LParametros.Caption:='(a,b,c)'
                  + #10 + #13 + 'a: ' + FloatTostr(PuntoQ[0])
                  + #10 + #13 + 'b: '+ FloatTostr(PuntoQ[1])
                  + #10 + #13 + 'c: '+ FloatTostr(PuntoQ[2]);

   end;

   if RBRecta.Checked=True then
   begin
      PuntoQ[0]:=StrToFloat(StringGrid1.Cells[0,1]);
      PuntoQ[1]:=StrToFloat(StringGrid1.Cells[1,1]);
```

```
    PuntoQ[2]:=StrToFloat(StringGrid1.Cells[2,1]);
    Puntop[0]:=StrToFloat(StringGrid1.Cells[0,2]);
    Puntop[1]:=StrToFloat(StringGrid1.Cells[1,2]);
    Puntop[2]:=StrToFloat(StringGrid1.Cells[2,2]);

    for i:=0 to 2 do
      u1[i]:= Puntop[i]-PuntoQ[i];

    normalizar(u1);

    for i:= 0 to 2 do
      begin
      ObjetoActual.Param[i]:= PuntoQ[i];
      ObjetoActual.Param[i+3]:= u1[i];
      end;

    ObjetoActual.obj:=ORecta;
    LObjeto.Caption:= 'Objeto: Recta';
    LParametros.Caption:='punto (a,b,c), vector (u,v,w)'
                + #10 + #13 + 'a: ' + FloatTostr(PuntoQ[0])
                + #10 + #13 + 'b: '+ FloatTostr(PuntoQ[1])
                + #10 + #13 + 'c: '+ FloatTostr(PuntoQ[2])
                + #10 + #13 + 'u: ' + FloatTostr(ObjetoActual.Param[3])
                 + #10 + #13 + 'v: '+ FloatTostr(ObjetoActual.Param[4])
                 + #10 + #13 + 'w: '+ FloatTostr(ObjetoActual.Param[5]);

 end;


if RBEsfera.Checked=True then
begin
   CalcularEsfera(B,npuntos,sol);
   ObjetoActual.obj:=Oesfera;
   {Calcular distancia maxima para tener una estimacion del error}
   error:=0;
   for i:=0 to 2 do
     centro[i]:=sol[i];

   for i:=0 to npuntos-1 do
     begin
```

```
      for j:=0 to 2 do
        puntop[j]:=B[i,j];

      dist:= abs(distanciaPuntoPunto(centro, puntop)-sol[3]);
    {  Showmessage('Punto: ' + #10 + #13 + 'a: ' + FloatTostr(puntop[0])
                + #10 + #13 + 'b: '+ FloatTostr(puntop[1])
                + #10 + #13 + 'c: '+ FloatTostr(puntop[2])
                + #10 + #13 + 'distancia: '+ FloatTostr(distanciaPuntoPunto(centro, puntop))); }
      if dist > error then error := dist;
    end;

  {fin de la estimacion}
  if RBIdeal.Checked=True then begin end
  else if RBExterna.Checked=True then sol[3]:=sol[3]-RPALPADOR
  else if RBinterna.Checked=True then sol[3]:=sol[3]+ RPALPADOR
  else if RBDisparo.Checked=True then
  begin
    PuntoQ[0]:=StrToFloat(StringGrid3.Cells[0,1]);
    PuntoQ[1]:=StrToFloat(StringGrid3.Cells[1,1]);
    PuntoQ[2]:=StrToFloat(StringGrid3.Cells[2,1]);
    centro[0]:=sol[0];
    centro[1]:=sol[1];
    centro[2]:=sol[2];
    if distanciaPuntoPunto(PuntoQ,Centro)>sol[3] then  sol[3]:=sol[3]-RPALPADOR
    else   sol[3]:=sol[3]+ RPALPADOR;
  end;

  for i:= 0 to 3 do
     ObjetoActual.Param[i]:= sol[i];
  LObjeto.Caption:= 'Objeto: Esfera';
  LParametros.Caption:='Centro (a,b,c), radio r'
                + #10 + #13 + 'a: ' + FloatTostr(sol[0])
                + #10 + #13 + 'b: '+ FloatTostr(sol[1])
                + #10 + #13 + 'c: '+ FloatTostr(sol[2])
                + #10 + #13 + 'r: '+ FloatTostr(sol[3])
                + #10 + #13 + 'error: '+ FloatTostr(error) ;
end;

if RBPlano.Checked=True then
begin
```

```
resolverplano(B,npuntos,v);
normalizar(v);
d:=terminoindep(B,npuntos,v);
punto[0]:=B[0,0];
punto[1]:=B[0,1];
punto[2]:=B[0,2];
puntodelplano(punto,v,d);

    {Calcular distancia maxima para tener una estimacion del error}
error:=0;

for i:=0 to npuntos-1 do
  begin
    for j:=0 to 2 do
      puntop[j]:=B[i,j];

    dist:= abs(distanciaPuntoPlano(puntop,punto,v));
  {  Showmessage('Punto: ' + #10 + #13 + 'a: ' + FloatTostr(puntop[0])
                + #10 + #13 + 'b: '+ FloatTostr(puntop[1])
                + #10 + #13 + 'c: '+ FloatTostr(puntop[2])
                + #10 + #13 + 'distancia: '+ FloatTostr(distanciaPuntoPunto(centro, puntop))); }
    if dist > error then error := dist;
  end;
  {Fin del calculo del error}

if RBIdeal.Checked=True then begin end
else if ((RBexterna.Checked=True) or (RBinterna.Checked=true))
   then showmessage('Se calcula el plano ideal')
else begin
{Almacenar el disparo al aire}
PuntoQ[0]:=StrToFloat(StringGrid3.Cells[0,1]);
PuntoQ[1]:=StrToFloat(StringGrid3.Cells[1,1]);
PuntoQ[2]:=StrToFloat(StringGrid3.Cells[2,1]);
{Aplicamos la traslación correspondiente}
if ((puntoq[0]-punto[0])*v[0] +(puntoq[1]-punto[1])*v[1] +(puntoq[2]-punto[2])*v[2]) > 0  then
begin
   punto[0]:=punto[0]-RPALPADOR*v[0];
   punto[1]:=punto[1]-RPALPADOR*v[1];
   punto[2]:=punto[2]-RPALPADOR*v[2];
```

```
  end
  else
  begin
    punto[0]:=punto[0]+RPALPADOR*v[0];
    punto[1]:=punto[1]+RPALPADOR*v[1];
    punto[2]:=punto[2]+RPALPADOR*v[2];
  end
  end;


  ObjetoActual.obj:=OPlano;
  for i:= 0 to 2 do
     ObjetoActual.Param[i]:= punto[i];
  for i:= 0 to 2 do
     ObjetoActual.Param[i+3]:= v[i];
  LObjeto.Caption:= 'Objeto: Plano';
  LParametros.Caption:='Punto (a,b,c), vector (u,v,w)'
                + #10 + #13 + 'a: ' + FloatTostr(punto[0])
                + #10 + #13 + 'b: '+ FloatTostr(punto[1])
                + #10 + #13 + 'c: '+ FloatTostr(punto[2])
                + #10 + #13 + 'u: ' + FloatTostr(v[0])
                + #10 + #13 + 'v: '+ FloatTostr(v[1])
                + #10 + #13 + 'w: '+ FloatTostr(v[2])
                + #10 + #13 + 'error: '+ FloatTostr(error)

end;

if RBCilindro.Checked=True then
begin

error:=CalcularCilindro(B,npuntos,sol1);
ObjetoActual.obj:=OCilindro;

if RBIdeal.Checked=True then begin end
else if RBExterna.Checked=True then sol1[2]:=sol1[2]-RPALPADOR
else if RBinterna.Checked=True then sol1[2]:=sol1[2]+ RPALPADOR
else if RBDisparo.Checked=True then
begin
  PuntoQ[0]:=StrToFloat(StringGrid3.Cells[0,1]);
  PuntoQ[1]:=StrToFloat(StringGrid3.Cells[1,1]);
```

```
  PuntoQ[2]:=0;
  centro[0]:=sol1[0];
  centro[1]:=sol1[1];
  centro[2]:=0;
  if distanciaPuntoPunto(PuntoQ,Centro)>sol1[2] then  sol1[2]:=sol1[2]-RPALPADOR
  else   sol1[2]:=sol1[2]+ RPALPADOR;
end;


for i:= 0 to 2 do
    ObjetoActual.Param[i]:= sol1[i];
LObjeto.Caption:= 'Objeto: Cilindro';
LParametros.Caption:='Centro (a,b), radio r'
                + #10 + #13 + 'a: ' + FloatTostr(sol1[0])
                + #10 + #13 + 'b: '+ FloatTostr(sol1[1])
                + #10 + #13 + 'r: '+ FloatTostr(sol1[2])
                + #10 + #13 + 'error: '+ FloatTostr(error)
 end;


if RBCono.Checked=True then
begin
{
{Desplazamos todos los puntos para que tengan coordenada z positiva y mayor que 1000}
minz:=B[0,2];
posminz:=0;
for i:=0 to npuntos-1 do
  if B[i,2]<minz then
  begin
    minz:=B[i,2];
    posminz:=i;
  end;
minz:=minz-1000;

for i:=0 to npuntos-1 do
    B[i,2]:=B[i,2]-minz;
}

   error:=CalcularConoMedias(B,npuntos,sol);
```

```
 {Showmessage('Centro ideal:' +FloatToStr(sol[0]) + #10 +#13
 +FloatToStr(sol[1]) + #10 +#13
 +FloatToStr(sol[2]) + #10 +#13
 +'Radio en z=0: '+FloatToStr(sol[3]) + #10 +#13
 ); }
  if RBIdeal.Checked=True then
  begin
  end
  else
  if RBExterna.Checked=True then
   begin
     if sol[2]>B[0,2] then sol[2]:=sol[2]-RPALPADOR*sqrt((sol[3]+1)/sol[3])
     else  sol[2]:=sol[2]+RPALPADOR*sqrt((sol[3]+1)/sol[3])
   end
   else if RBinterna.Checked=True then
   begin
     if sol[2]>B[0,2] then sol[2]:=sol[2]+RPALPADOR*sqrt((sol[3]+1)/sol[3])
     else  sol[2]:=sol[2]-RPALPADOR*sqrt((sol[3]+1)/sol[3])
   end
   else if RBDisparo.Checked=True then
   begin
   {Calculamos donde esta el punto, si da negativo, esta dentro del cono, si da positivo, fuera}
    PuntoQ[0]:=StrToFloat(StringGrid3.Cells[0,1]);
    PuntoQ[1]:=StrToFloat(StringGrid3.Cells[1,1]);
    PuntoQ[2]:=StrToFloat(StringGrid3.Cells[2,1])+minz;
    distanciaACono:=sqr(PuntoQ[0]-sol[0])+sqr(PuntoQ[1]-sol[1])-sol[3]*sqr(PuntoQ[2]-sol[2]);
 if distanciaACono<0 then
 begin
   {Posicion interna, hacemos como antes}
   if sol[2]>B[0,2] then sol[2]:=sol[2]+RPALPADOR*sqrt((sol[3]+1)/sol[3])
   else  sol[2]:=sol[2]-RPALPADOR*sqrt((sol[3]+1)/sol[3])
 end
 else
 begin
   {Posicion externa}
   if sol[2]>B[0,2] then sol[2]:=sol[2]-RPALPADOR*sqrt((sol[3]+1)/sol[3])
   else  sol[2]:=sol[2]+RPALPADOR*sqrt((sol[3]+1)/sol[3])
 end;

end;
```

```
    {
   {Restituimos la altura de la coordenada z para la solucion}

   sol[2]:=sol[2]+minz;

   {El radio del cono esta calculado a una altura +1000. por lo tanto hay que
   hacer ahora el ajuste necesario para restituirlo}

   sol[3]:=(sol[2])/(sol[2]-minz)*sol[3];
   }

   ObjetoActual.obj:=Ocono;
   for i:= 0 to 3 do
      ObjetoActual.Param[i]:= sol[i];
   LObjeto.Caption:= 'Objeto: Cono';
   LParametros.Caption:='Centro (a,b,c), radio r'
                    + #10 + #13 + 'a: ' + FloatTostr(sol[0])
                    + #10 + #13 + 'b: '+ FloatTostr(sol[1])
                    + #10 + #13 + 'c: '+ FloatTostr(sol[2])
                    + #10 + #13 + 'r: '+ FloatTostr(sol[3])
                    + #10 + #13 + 'error: '+ FloatTostr(error) ;
end;


{Cilindro girado. Es copia del cilindro en muchos aspectos}

if RBCilindroGirado.Checked=True then
begin

{A partir del plano obtenemos su vector normal}
resolverplano(B2,npuntos2,u1);
{Obtenemos la matriz de giro}
matrizGiroVector(matA,u1);
error:=resolvercilindrogirado(B, npuntos, matA, u1, radio, c);
{Ahora hay un punto del eje en el vector c, el radio en r, y en u1 esta el vector direccion}

{Acaba}
ObjetoActual.obj:=OCilindroGirado;
```

```
if RBIdeal.Checked=True then begin end
else if RBExterna.Checked=True then radio:=radio-RPALPADOR
else if RBinterna.Checked=True then radio:=radio+ RPALPADOR
else if RBDisparo.Checked=True then
begin
  PuntoQ[0]:=StrToFloat(StringGrid3.Cells[0,1]);
  PuntoQ[1]:=StrToFloat(StringGrid3.Cells[1,1]);
  PuntoQ[2]:=StrToFloat(StringGrid3.Cells[2,1]);
  centro[0]:=c[0];
  centro[1]:=c[1];
  centro[2]:=c[2];
  if distanciaPuntoPunto(PuntoQ,Centro)>radio then  radio:=radio-RPALPADOR
  else   radio:=radio+ RPALPADOR;
end;



for i:= 0 to 2 do
    ObjetoActual.Param[i]:= c[i];
ObjetoActual.Param[3]:=radio;
for i:= 4 to 6 do
    ObjetoActual.Param[i]:= u1[i-4];

LObjeto.Caption:= 'Objeto: Cilindro girado';
LParametros.Caption:='Centro (a,b,c), radio r, direccion (u,v,w)'
                + #10 + #13 + 'a: ' + FloatTostr(c[0])
                + #10 + #13 + 'b: '+ FloatTostr(c[1])
                + #10 + #13 + 'c: ' + FloatTostr(c[2])
                + #10 + #13 + 'r: '+ FloatTostr(radio)
                + #10 + #13 + 'u: ' + FloatTostr(u1[0])
                + #10 + #13 + 'v: '+ FloatTostr(u1[1])
                + #10 + #13 + 'w: '+ FloatTostr(u1[2])
                + #10 + #13 + 'error: '+ FloatTostr(error)
 end;

{Cono girado. Es copia del cilindro en muchos aspectos}

if RBConoGirado.Checked=True then
begin

{Comienza el pegado}
```

```
  resolverplano(B2,npuntos2,u1);
  {Obtenemos la matriz de giro}
  matrizGiroVector(matA,u1);
{Ahora hacemos lo mismo que para el cono}
{**************************************}
{invertir trasponiendo porque es simetrica}
for i:=0 to 2 do
   for j:=0 to 2 do
      Ainv[i,j]:=matA[j,i];

for i:=0 to (npuntos-1) do
   begin
      for j:=0 to 2 do
         bi[j]:=B[i,j];

      multmatrizvector(Ainv,Bi,Bibis);

      for j:=0 to 2 do
         B[i,j]:=bibis[j];

   end;

{MostrarMatriz20x3(B,npuntos);   }

{
{Desplazamos todos los puntos para que tengan coordenada z positiva y mayor que 1000}
minz:=B[0,2];
posminz:=0;
for i:=0 to npuntos-1 do
  if B[i,2]<minz then
  begin
    minz:=B[i,2];
    posminz:=i;
  end;
minz:=minz-1000;

for i:=0 to npuntos-1 do
    B[i,2]:=B[i,2]-minz;
 }
```

```
 error:=CalcularConoMedias(B,npuntos,sol);

   if RBIdeal.Checked=True then
   begin end
   else if RBExterna.Checked=True then
   begin
     if sol[2]>B[0,2] then sol[2]:=sol[2]-RPALPADOR*sqrt((sol[3]+1)/sol[3])
     else  sol[2]:=sol[2]+RPALPADOR*sqrt((sol[3]+1)/sol[3])
   end
   else if RBinterna.Checked=True then
   begin
     if sol[2]>B[0,2] then sol[2]:=sol[2]+RPALPADOR*sqrt((sol[3]+1)/sol[3])
     else  sol[2]:=sol[2]-RPALPADOR*sqrt((sol[3]+1)/sol[3])
   end
   else if RBDisparo.Checked=True then
   begin
   {Calculamos donde esta el punto, si da negativo, esta dentro del cono, si da positivo, fuera}
    PuntoQbis[0]:=StrToFloat(StringGrid3.Cells[0,1]);
    PuntoQbis[1]:=StrToFloat(StringGrid3.Cells[1,1]);
    PuntoQbis[2]:=StrToFloat(StringGrid3.Cells[2,1]);
    multmatrizvector(Ainv,PuntoQbis,PuntoQ);
    PuntoQ[2]:=PuntoQ[2]-minz;
    distanciaACono:=sqr(PuntoQ[0]-sol[0])+sqr(PuntoQ[1]-sol[1])-sol[3]*sqr(PuntoQ[2]-sol[2]);
  if distanciaACono<0 then
  begin
    {Posicion interna, hacemos como antes}
    if sol[2]>B[0,2] then sol[2]:=sol[2]+RPALPADOR*sqrt((sol[3]+1)/sol[3])
    else  sol[2]:=sol[2]-RPALPADOR*sqrt((sol[3]+1)/sol[3])
  end
  else
  begin
    {Posicion externa}
   if sol[2]>B[0,2] then sol[2]:=sol[2]-RPALPADOR*sqrt((sol[3]+1)/sol[3])
   else  sol[2]:=sol[2]+RPALPADOR*sqrt((sol[3]+1)/sol[3])
  end;

end;

    {
    {Restituimos la altura de la coordenada z para la solucion}
```

```
 sol[2]:=sol[2]+minz;
  }

 ObjetoActual.obj:=OconoGirado;
 for i:=0 to 2 do
   sol1[i]:=sol[i];

 MultmatrizVector(matA,sol1,sol1bis);
 for i:= 0 to 2 do
    ObjetoActual.Param[i]:= sol1bis[i];

 objetoActual.Param[3]:=sol[3];

 for i:= 4 to 6 do
    ObjetoActual.Param[i]:= u1[i-4];


 LObjeto.Caption:= 'Objeto: Cono Girado';
 LParametros.Caption:='Centro (a,b,c), radio r, vector (u,v,w)'
                + #10 + #13 + 'a: '+ FloatTostr(sol1bis[0])
                + #10 + #13 + 'b: '+ FloatTostr(sol1bis[1])
                + #10 + #13 + 'c: '+ FloatTostr(sol1bis[2])
                + #10 + #13 + 'r: '+ FloatTostr(sol[3])
                + #10 + #13 + 'u: '+ FloatTostr(u1[0])
                + #10 + #13 + 'v: '+ FloatTostr(u1[1])
                + #10 + #13 + 'w: '+ FloatTostr(u1[2])
                + #10 + #13 + 'error: '+ FloatTostr(error);

end;
{*************************************}

 Banadir.Enabled:=True;
 except
   On E: Exception do
   Showmessage('Error al convertir los puntos o al calcular el objeto')

 end;
```

```
end;

procedure TForm1.BCalcular2Click(Sender: TObject);
var
  nobjeto: integer;
begin
 nobjeto:=StrToInt(Edit1.Caption);
 if (nobjeto >= 0) and (nobjeto < nobjetos) then
 begin
 if ((ListaObjetos[nobjeto].obj= Ocono) or (ListaObjetos[nobjeto].obj= OconoGirado)) then
 begin
   puntoActual.obj:=OPunto;
   puntoActual.param[0]:=ListaObjetos[nobjeto].param[0];
   puntoActual.param[1]:=ListaObjetos[nobjeto].param[1];
   puntoActual.param[2]:=ListaObjetos[nobjeto].param[2];
   BAnadir2.Enabled:=True;
   LParametros1.Caption:='a = ' + FloatToStr(puntoActual.param[0]) +#10 + #13 +
                         'b = ' + FloatToStr(puntoActual.param[1]) +#10 + #13 +
                         'c = ' + FloatToStr(puntoActual.param[2]) +#10 + #13 +
                         'Ángulo = ' + FloatToStr(Arctan(sqrt(ListaObjetos[nobjeto].param[3])));

 end;

 if ListaObjetos[nobjeto].obj= Oesfera then
 begin
   puntoActual.obj:=OPunto;
   puntoActual.param[0]:=ListaObjetos[nobjeto].param[0];
   puntoActual.param[1]:=ListaObjetos[nobjeto].param[1];
   puntoActual.param[2]:=ListaObjetos[nobjeto].param[2];
   BAnadir2.Enabled:=True;
   LParametros1.Caption:='a = ' + FloatToStr(puntoActual.param[0]) +#10 + #13 +
                         'b = ' + FloatToStr(puntoActual.param[1]) +#10 + #13 +
                         'c = ' + FloatToStr(puntoActual.param[2]) +#10 + #13;
 end;

   if not( (ListaObjetos[nobjeto].obj= Ocono) or ( ListaObjetos[nobjeto].obj= Oesfera) or (ListaObjetos[nobjeto].obj=
OconoGirado))then
 begin
   ShowMessage('Solo puede aplicarse a esferas o conos')
```

```
 end;

 end
 else showmessage('Número de objeto inválido') ;


end;

procedure TForm1.Button1Click(Sender: TObject);
var
  objeto1,objeto2,i: integer;
  distancia, distancia2,t,D: real;
  v1,v2,w1,w2,punto1,punto2,punto3,v:tvector3;
  calculado: boolean;
  cadena: String;
begin
  calculado:=false;
  objeto1:=StrToInt(Edit4.Text);
  objeto2:=StrToInt(Edit5.Text);
  if (objeto1 >= 0) and (objeto1<nobjetos) and (objeto2 >= 0) and (objeto2<nobjetos)
  then
  begin
        {Almacenamos un punto de cada objeto}
  for i:= 0 to 2 do
    begin
      punto1[i]:=ListaObjetos[Objeto1].Param[i];
      punto2[i]:=ListaObjetos[Objeto2].Param[i];
    end;
  {showmessage('v ' + FloatTostr(punto1[0]) +#10+#13
               + FloatTostr(punto1[1]) +#10+#13
               + FloatTostr(punto1[2]) +#10+#13
               + FloatTostr(punto2[0]) +#10+#13
               + FloatTostr(punto2[1]) +#10+#13
               + FloatTostr(punto2[2]) +#10+#13 );   }

      if ((ListaObjetos[objeto1].obj = OPlano) or (ListaObjetos[objeto1].obj = ORecta))
       then
       for i:= 0 to 2 do
          v1[i]:=ListaObjetos[Objeto1].Param[i+3];
```

```
 if ((ListaObjetos[objeto2].obj = OPlano) or (ListaObjetos[objeto2].obj = ORecta))
  then
      for i:= 0 to 2 do
         v2[i]:=ListaObjetos[Objeto2].Param[i+3];
{       showmessage('v ' + FloatTostr(v1[0]) +#10+#13
                + FloatTostr(v1[1]) +#10+#13
                + FloatTostr(v1[2]) +#10+#13
                + FloatTostr(v2[0]) +#10+#13
                + FloatTostr(v2[1]) +#10+#13
                + FloatTostr(v2[2]) +#10+#13 );  }

 {Interseccion de dos planos}

 if ((ListaObjetos[objeto1].obj = OPlano) and (ListaObjetos[objeto2].obj = OPlano))
 then
 begin
 calculado:= puntodosplanos(punto1,v1,punto2,v2,punto3);
 showmessage('Punto' + FloatTostr(punto3[0]) + #10 +#13 +
                  FloatTostr(punto3[1]) + #10 +#13 +
                  FloatTostr(punto3[2]) + #10 +#13);

 if calculado=true then
 begin
  v[0]:=v1[1]*v2[2]-v1[2]*v2[1];
  v[1]:=v1[2]*v2[0]-v1[0]*v2[2];
  v[2]:=v1[0]*v2[1]-v1[1]*v2[0];
  if norma(v)>0 then calculado:=true else calculado:=false;
 end;

 if calculado=true then
 begin
  InterseccionActual.Obj:=ORecta;
  normalizar(v);
  for i:=0 to 2 do
   begin
     InterseccionActual.Param[i]:= punto3[i];
     InterseccionActual.Param[i+3]:=v[i];
   end;
```

```
  Button2.Enabled:=true;
  Label5.Caption:= 'Recta con punto (a,b,c) y vector (u,v,w)'+#10+#13+
                   'a: ' + FloatTostr(InterseccionActual.Param[0]) +#10+#13+
                   'b: ' + FloatTostr(InterseccionActual.Param[1]) +#10+#13+
                   'c: ' + FloatTostr(InterseccionActual.Param[2]) +#10+#13+
                   'u: ' + FloatTostr(InterseccionActual.Param[3]) +#10+#13+
                   'v: ' + FloatTostr(InterseccionActual.Param[4]) +#10+#13+
                   'w: ' + FloatTostr(InterseccionActual.Param[5]);
  Showmessage('Ángulo entre planos: ' +
              FloatToStr(ArcCos(abs(v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2])/(norma(v1)*norma(v2)))));
end
else showmessage('Error: planos paralelos');

end;

{Fin interseccion de dos planos}

{Interseccion recta-plano}


    if ((ListaObjetos[objeto1].obj = ORecta) and (ListaObjetos[objeto2].obj = OPlano))
    then
    begin
    D:=-(punto2[0]*v2[0]+ punto2[1]*v2[1]+punto2[2]*v2[2]);
    if (v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2]) = 0 then showmessage('Error: recta paralela a plano')
    else
    begin
    t:=(-D-v2[0]*punto1[0]-v2[1]*punto1[1]-v2[2]*punto1[2])/(v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2]);
    for i:=0 to 2 do
      punto3[i]:=punto1[i]+t*v1[i];
    calculado:=true;

      InterseccionActual.Obj:=OPunto;
      for i:=0 to 2 do
        begin
          InterseccionActual.Param[i]:= punto3[i];
        end;
      Button2.Enabled:=true;
      Label5.Caption:= 'Punto (a,b,c)'+#10+#13+
                       'a: ' + FloatTostr(InterseccionActual.Param[0]) +#10+#13+
```

```
            'b: ' + FloatTostr(InterseccionActual.Param[1]) +#10+#13+
            'c: ' + FloatTostr(InterseccionActual.Param[2]);
  Showmessage('Ángulo entre recta y normal al plano: ' +
          FloatToStr(ArcCos(abs(v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2])/(norma(v1)*norma(v2)))));

end;

end;



{Interseccion plano-recta}


    if ((ListaObjetos[objeto1].obj = OPlano) and (ListaObjetos[objeto2].obj = Orecta))
    then
    begin
    D:=-(punto1[0]*v1[0]+ punto1[1]*v1[1]+punto1[2]*v1[2]);
    if (v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2]) = 0 then showmessage('Error: recta paralela a plano')
    else
    begin
    t:=(-D-v1[0]*punto2[0]-v1[1]*punto2[1]-v1[2]*punto2[2])/(v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2]);
    for i:=0 to 2 do
      punto3[i]:=punto2[i]+t*v2[i];
    calculado:=true;

      InterseccionActual.Obj:=OPunto;
      for i:=0 to 2 do
        begin
          InterseccionActual.Param[i]:= punto3[i];
        end;
      Button2.Enabled:=true;
      Label5.Caption:= 'Punto (a,b,c)'+#10+#13+
                  'a: ' + FloatTostr(InterseccionActual.Param[0]) +#10+#13+
                  'b: ' + FloatTostr(InterseccionActual.Param[1]) +#10+#13+
                  'c: ' + FloatTostr(InterseccionActual.Param[2]);
      Showmessage('Ángulo entre recta y normal al plano: ' +
              FloatToStr(ArcCos(abs(v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2])/(norma(v1)*norma(v2)))));

    end;
```

```
            end;


            {Angulo recta-recta}


                if ((ListaObjetos[objeto1].obj = ORecta) and (ListaObjetos[objeto2].obj = Orecta))
                then
                begin

                  Showmessage('Ángulo entre rectas: ' +
                          FloatToStr(ArcCos(abs(v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2])/(norma(v1)*norma(v2)))));


                end;



  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  cadena: string;
begin

  ListaObjetos[nobjetos]:=InterseccionActual;
  nobjetos:=nobjetos+1;
  SGListaObjetos.Cells[0,nobjetos]:= IntToStr(nobjetos-1);

  SGListaObjetos.Cells[2,nobjetos]:= FloatToStr(InterseccionActual.Param[0]);
  SGListaObjetos.Cells[3,nobjetos]:= FloatToStr(InterseccionActual.Param[1]);
  SGListaObjetos.Cells[4,nobjetos]:= FloatToStr(InterseccionActual.Param[2]);

  if interseccionActual.Obj=ORecta then
  begin
    SGListaObjetos.Cells[1,nobjetos]:= 'Recta';
    SGListaObjetos.Cells[5,nobjetos]:= FloatToStr(InterseccionActual.Param[3]);
    SGListaObjetos.Cells[6,nobjetos]:= FloatToStr(InterseccionActual.Param[4]);
```

```
    SGListaObjetos.Cells[7,nobjetos]:= FloatToStr(InterseccionActual.Param[5]);
  end;

  if interseccionActual.Obj=OPunto then
  begin
    SGListaObjetos.Cells[1,nobjetos]:= 'Punto';
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  i,j:integer;
begin
  for i:=1 to StringGrid1.RowCount-1 do
    for j:=0 to StringGrid1.ColCount-1 do
      StringGrid1.Cells[j,i]:='';

end;

procedure TForm1.Button4Click(Sender: TObject);
var
  i,j:integer;
begin
  for i:=1 to StringGrid2.RowCount-1 do
    for j:=0 to StringGrid2.ColCount-1 do
      StringGrid2.Cells[j,i]:='';

end;

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: char);
begin
   if not(Key in ['0'..'9', #8]) then
 begin
 key := #0;
   beep;
end;
end;

procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: char);
begin
```

```
  if not(Key in ['0'..'9', #8]) then
 begin
 key := #0;
  beep;
end;
end;

procedure TForm1.Edit3KeyPress(Sender: TObject; var Key: char);
begin
  if not(Key in ['0'..'9', #8]) then
 begin
 key := #0;
  beep;
end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin

end;

procedure TForm1.LObjeto1Click(Sender: TObject);
begin

end;

procedure TForm1.StringGrid1KeyPress(Sender: TObject; var Key: char);
begin
  if not(Key in ['0'..'9', #8, ',', '-']) then
 begin
 key := #0;
  beep;
end;
end;

procedure TForm1.StringGrid2KeyPress(Sender: TObject; var Key: char);
begin
  if not(Key in ['0'..'9', #8, ',', '-']) then
 begin
 key := #0;
```

```
   beep;
end;
end;

procedure TForm1.StringGrid3KeyPress(Sender: TObject; var Key: char);
begin
  if not(Key in ['0'..'9', #8, ',', '-']) then
 begin
 key := #0;
   beep;
  end;
end;


end.
```

```
unit mates;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls;
type
   tmatriz2x2 = array[0..2,0..2] of real;
   tmatriz3x3 = array[0..2,0..2] of real;
   tmatriz4x3 = array[0..3,0..2] of real;
   tmatriz4x4 = array[0..3,0..3] of real;
   tmatriz5x5 = array[0..4,0..4] of real;
   tmatriz5x3 = array[0..4,0..2] of real;
   tmatriz20x3 = array[0..19,0..2] of real;
   tmatriz20x2 = array[0..19,0..1] of real;
   tmatriz20x4 = array[0..19,0..3] of real;
   tvector2 = array[0..1] of real;
   tvector3 = array[0..2] of real;
   tvector4 = array[0..3] of real;
   tvector5 = array[0..4] of real;
   tvectorn = array[0..19] of real;




   function arccos(x:real):real;
   function norma(var b: tvector4):real;
   function norma(var b: tvector3):real;
   function norma(var b: tvector5):real;
   function distanciaPuntoPunto(var a,b: tvector3):real;
   function distanciaPuntoPlano(var a,b,n: tvector3):real;
   function distanciaPuntoRecta(var a,b,n: tvector3):real;
   function distanciaRectaRecta(var a,b,n,m: tvector3):real;
   function puntodosplanos (var P1,v1,P2,v2,P3:tvector3):boolean;
   procedure MatrizGiroVector(var matA:tmatriz3x3; var u1:tvector3);

   procedure MostrarMatriz3x3(var A:tmatriz3x3);
   procedure MostrarMatriz20x3(var B:tmatriz20x3; npuntos:integer);
```

```
  procedure MostrarVector3(var v:tvector3);
  procedure normalizar(var b: tvector3);

  function det2 (var A:tmatriz2x2): real;
  function determinante (var A:tmatriz3x3): real;
  function determinante (var A:tmatriz4x4): real;
  function determinante (var A:tmatriz5x5): real;

  function distanciapuntos (var a:tvector3; var b: tvector3): real;

  procedure resolver (var A:tmatriz2x2; var v:tvector2; var b: tvector2);
  function resolver (var A:tmatriz3x3; var v:tvector3; var b: tvector3):boolean;
  function resolver (var A:tmatriz4x4; var v:tvector4; var b: tvector4):boolean;
  procedure resolver (var A:tmatriz5x5; var v:tvector5; var b: tvector5);

  procedure invertirmatriz(var A: tmatriz3x3; var Ainversa: tmatriz3x3);

  procedure prepararmatrizplano(var P: tmatriz20x3; npuntos:integer; var A: tmatriz3x3; var b:tvector3);
  procedure resolverplano (var P: tmatriz20x3; npuntos: integer; var sol: tvector3);

  {Esfera}
  procedure esferac (var A:tmatriz4x3; var centro:tvector3; var radio: real);
  function NResferac (var P:tmatriz20x3; npuntos: integer ; var sol:tvector4):integer;
  function CalcularEsfera (var B:tmatriz20x3; npuntos: integer ; var sol:tvector4):integer;

  {Cilindro}
  function CalcularCilindro (var B:tmatriz20x3; npuntos: integer ; var sol:tvector3):real;
  procedure cilindroc (var A:tmatriz3x3; var centro:tvector2; var radio: real);
  procedure fun (var P:tmatriz20x3; npuntos: integer ; var sol:tvector4; var funcion: tvector4);
  procedure jac (var P:tmatriz20x3; npuntos: integer ; var sol:tvector4; var jacobiano:tmatriz4x4);

    function resolvercilindrogirado(var B: tmatriz20x3; npuntos: integer; var A: tmatriz3x3; var u1: tvector3; var
radio:real; var c: tvector3): real;

  procedure conoc(var P:tmatriz20x3; npuntos: integer; var sol:tvector4);
  procedure conoc5puntos(var P:tmatriz5x3; var sol:tvector4);

  function Mcilindroc (var P:tmatriz20x3; npuntos: integer ; var sol:tvector3):real;
```

```
function NRcilindroc (var P:tmatriz20x3; npuntos: integer ; var sol:tvector3):integer;
procedure funCilindro (var P:tmatriz20x3; npuntos: integer ; var sol:tvector3; var funcion: tvector3);
procedure jacCilindro (var P:tmatriz20x3; npuntos: integer ; var sol:tvector3; var jacobiano:tmatriz3x3);

function NRcilindroGiradoc (var P:tmatriz20x3; npuntos: integer ; var sol:tvector5):integer;
procedure funCilindroGirado (var P:tmatriz20x3; npuntos: integer ; var sol:tvector5; var funcion: tvector5);
procedure jacCilindroGirado (var P:tmatriz20x3; npuntos: integer ; var sol:tvector5; var jacobiano:tmatriz5x5);

function G (n: integer; var X: tmatriz20x3; var w: tvector3; var pc: tvector3; var rsqr: real): real;
procedure OuterProduct(var w: tvector3; var wwt: tmatriz3x3);
function Dot(var A: tvector3; var B: tvector3): real;
procedure MultMatrizVector(var A: tmatriz3x3; var b: tvector3; var c: tvector3);
procedure MultVectorMatriz(var b: tvector3; var A: tmatriz3x3;  var c: tvector3);
procedure SumMatrices(var A: tmatriz3x3; var B: tmatriz3x3; var C: tmatriz3x3);
procedure AsigMatrices(var A: tmatriz3x3; var B: tmatriz3x3);
function FitCylinder (n: integer; var X: tmatriz20x3; var rsqr: real; var C: tvector3; var W: tvector3): real;

{Calcular cilindro medias}
function CalcularCilindroMedias (var B:tmatriz20x3; npuntos: integer ; var sol:tvector3):real;

{NR para el cono a partir de 4 puntos}
procedure funcono (var P:tmatriz4x3; var sol:tvector4; var funcion: tvector4);
procedure jaccono (var P:tmatriz4x3; var sol:tvector4; var jac: tmatriz4x4);
function NRconoc (var P:tmatriz4x3; var sol:tvector4):integer;

{NR para el cono recto con npuntos }
procedure funcono (var P:tmatriz20x3; npuntos: integer ; var sol:tvector4; var funcion: tvector4);
procedure jaccono (var P:tmatriz20x3; npuntos: integer ; var sol:tvector4; var jac: tmatriz4x4);
function NRconoc (var P:tmatriz20x3; npuntos: integer ; var sol:tvector4):integer;

{Cono con 5 puntos}
procedure eliminarcd (var A:tmatriz4x4; var indep: tvector4; var A2:tmatriz3x3; var indep2: tvector3);
procedure resolversistemacono(neq: integer; var A: tmatriz20x4; var sol: tvector4; var indep: tvectorn);

{Calcular cono}
function CalcularCono (var B:tmatriz20x3; npuntos: integer ; var sol:tvector4):integer;
function CalcularConoMedias (var B:tmatriz20x3; npuntos: integer ; var sol:tvector4):real;

{Termino independiente del plano a partir de la lista de puntos}
function terminoindep (var B:tmatriz20x3; npuntos: integer ; var sol:tvector3):real;
```

```
  {Obtener el primer punto del plano a partir de un punto cercano, el vector normal y el termino indep
  el punto P contiene la aproximación al punto y devuelve el punto calculado}
  procedure puntodelplano(var P:tvector3; var sol:tvector3; d: real);

  const

  RPALPADOR = 1;

implementation
procedure MostrarMatriz20x3(var B:tmatriz20x3; npuntos:integer);
var
 i,j:integer;
 cadena:string;
begin
{Mostramos por pantalla la matriz B}

cadena:='';
for i:=0 to npuntos-1 do
begin
for j:=0 to 2 do
begin
  cadena:=cadena + FloatToStr(B[i,j]);
   cadena:= cadena + '   ';
end;
cadena:= cadena + #10 + #13;
end;

showmessage(cadena);



end;

{Obtiene la matriz de giro a partir de un vector del eje. Modifica los dos parametros}
procedure MatrizGiroVector(var matA:tmatriz3x3; var u1:tvector3);
var
  u2,u3:tvector3;
  i:integer;

begin
  if not ((u1[0]=0) and (u1[1]=0)) then
```

```
begin

u2[0]:=-u1[1];
u2[1]:=u1[0];
u2[2]:=0;



u3[0]:=-u1[0]* u1[2];
u3[1]:=-u1[1]* u1[2];
u3[2]:=sqr(u1[0]) +  sqr(u1[1]);

end else
begin
u2[0]:=0;
u2[1]:=-u1[2];
u2[2]:=u1[1];



u3[0]:=sqr(u1[1]) +  sqr(u1[2]);
u3[1]:=-u1[0]* u1[1];
u3[2]:=-u1[0]* u1[2];
end;
{  showmessage('vector u1 : ' + FloatTostr(u1[0]) +
  FloatTostr(u1[1]) +
  FloatTostr(u1[2]) + #10 + #13 +
  'vector u2 : ' + FloatTostr(u2[0]) +
  FloatTostr(u2[1]) +
  FloatTostr(u2[2]) + #10 + #13 +
  'vector u3 : ' + FloatTostr(u3[0]) +
  FloatTostr(u3[1]) +
  FloatTostr(u3[2]));   }
normalizar(u1);
{ showmessage('vector u1');   }
normalizar(u2);
{showmessage('vector u1');  }
normalizar(u3);
{showmessage('vector u1'); }

for i:=0 to 2 do
  begin
```

```
      matA[i,0]:=u2[i];
      matA[i,1]:=u3[i];
      matA[i,2]:=u1[i];
    end;
   {MostrarMatriz3x3(matA); }
end;

procedure MostrarVector3(var v:tvector3);
var
  i,j:integer;
  cadena:string;

  begin
    cadena:='';
    for i:=0 to 2 do
    begin
      cadena:=cadena+FloatTostr(v[i]) + '   ';
      cadena:=cadena + #10 + #13;
    end;
    showmessage('Mostrar vector: ' + cadena);

  end;

procedure MostrarMatriz3x3(var A:tmatriz3x3);
var
  i,j:integer;
  cadena:string;
begin
  cadena:='';
  for i:=0 to 2 do
  begin
    for j:=0 to 2 do
    begin
      cadena:=cadena+FloatTostr(A[i,j]) + '   ';
    end;
    cadena:=cadena + #10 + #13;
  end;
  showmessage('Mostrar matriz: '+cadena);

end;
```

```
function distanciapuntos (var a:tvector3; var b: tvector3): real;
begin
  distanciapuntos:=sqrt(sqr(a[0]-b[0])+sqr(a[1]-b[1])+sqr(a[2]-b[2]));
end;


procedure prepararmatrizplano(var P: tmatriz20x3; npuntos:integer; var A: tmatriz3x3; var b:tvector3);
var
  i,j: integer;
begin

  for i:=0 to 2 do
    begin
    b[i]:=0;
    for j:=0 to 2 do
      A[i,j]:=0;

    end;

  for i:=0 to (npuntos -1) do
    begin
      A[0,0]:=A[0,0] + sqr(P[i,0]);
      A[0,1]:=A[0,1] + P[i,0]*P[i,1];
      A[0,2]:=A[0,2] + P[i,0]*P[i,2];

      A[1,1]:=A[1,1] + sqr(P[i,1]);
      A[1,2]:=A[1,2] + P[i,1]*P[i,2];

      A[2,2]:=A[2,2] + sqr(P[i,2]);

      b[0]:=b[0]+P[i,0];
      b[1]:=b[1]+P[i,1];
      b[2]:=b[2]+P[i,2];

    end;

  A[1,0]:=A[0,1];
  A[2,0]:=A[0,2];
  A[2,1]:=A[1,2];
```

```
end;

procedure resolverplano (var P: tmatriz20x3; npuntos: integer; var sol: tvector3);
var
  i,j: integer;
  A: tmatriz3x3;
  b:tvector3;
  cadena: String;
begin

  cadena:='';
  for i:=0 to npuntos-1 do
    begin
    for j:=0 to 2 do
      begin
        cadena:=cadena + ' ' +  FloatToStr(P[i,j]);
      end;
    cadena:=cadena + #10 + #13;
    end;

  { showmessage('En resolver plano. Matriz de puntos' + #10 +#13 + cadena);}
  prepararmatrizplano(P,npuntos,A,b);

  if not (resolver(A,sol,b)) then
  begin
    for i:=0 to npuntos do
      P[i,0]:=P[i,0]+1;
      prepararmatrizplano(P,npuntos,A,b);
    for i:=0 to npuntos do
      P[i,0]:=P[i,0]-1;
    if not (resolver(A,sol,b)) then
    begin
      for i:=0 to npuntos do
        P[i,1]:=P[i,1]+1;
        prepararmatrizplano(P,npuntos,A,b);
      for i:=0 to npuntos do
        P[i,1]:=P[i,1]-1;
      if not (resolver(A,sol,b)) then
```

```
      begin
        for i:=0 to npuntos do
           P[i,2]:=P[i,2]+1;
           prepararmatrizplano(P,npuntos,A,b);
        for i:=0 to npuntos do
           P[i,2]:=P[i,2]-1;
        if not (resolver(A,sol,b))  then showmessage('Error')
      end;
    end;
  end;

  { showmessage('En resolver plano vector sol : ' +#10 +#13 +
  FloatTostr(sol[0]) + #10 +#13 +
  FloatTostr(sol[1]) + #10 +#13 +
  FloatTostr(sol[2])
  );  }

end;

procedure normalizar(var b: tvector3);
var
  norm: real;
begin
  norm:=norma(b);
  b[0]:=b[0]/norm;
  b[1]:=b[1]/norm;
  b[2]:=b[2]/norm;
end;

function determinante (var A:tmatriz3x3): real;
begin
 determinante:=A[0,0]*A[1,1]*A[2,2] + A[0,1]*A[1,2]*A[2,0]
 + A[0,2]*A[1,0]*A[2,1] - A[0,2]*A[1,1]*A[2,0] -A[0,1]*A[1,0]*A[2,2]
 -A[0,0]*A[1,2]*A[2,1];
end;

function determinante (var A:tmatriz4x4): real;
var
  subA: tmatriz3x3;
  i, j, k, signo: integer;
```

```
  acumulador,ad0,ad1,ad2,ad3: real;

begin
  acumulador:=0;
  signo:=1;
  {ad0}
  subA[0,0]:=A[1,1];
  subA[0,1]:=A[1,2];
  subA[0,2]:=A[1,3];
  subA[1,0]:=A[2,1];
  subA[1,1]:=A[2,2];
  subA[1,2]:=A[2,3];
  subA[2,0]:=A[3,1];
  subA[2,1]:=A[3,2];
  subA[2,2]:=A[3,3];
  ad0:=determinante(subA);

    {ad1}
  subA[0,0]:=A[1,0];
  subA[0,1]:=A[1,2];
  subA[0,2]:=A[1,3];
  subA[1,0]:=A[2,0];
  subA[1,1]:=A[2,2];
  subA[1,2]:=A[2,3];
  subA[2,0]:=A[3,0];
  subA[2,1]:=A[3,2];
  subA[2,2]:=A[3,3];
  ad1:=determinante(subA);

    {ad2}
  subA[0,0]:=A[1,0];
  subA[0,1]:=A[1,1];
  subA[0,2]:=A[1,3];
  subA[1,0]:=A[2,0];
  subA[1,1]:=A[2,1];
  subA[1,2]:=A[2,3];
  subA[2,0]:=A[3,0];
  subA[2,1]:=A[3,1];
  subA[2,2]:=A[3,3];
  ad2:=determinante(subA);
```

```
    {ad3}
  subA[0,0]:=A[1,0];
  subA[0,1]:=A[1,1];
  subA[0,2]:=A[1,2];
  subA[1,0]:=A[2,0];
  subA[1,1]:=A[2,1];
  subA[1,2]:=A[2,2];
  subA[2,0]:=A[3,0];
  subA[2,1]:=A[3,1];
  subA[2,2]:=A[3,2];
  ad3:=determinante(subA);




  determinante:=A[0,0]*ad0-A[0,1]*ad1+A[0,2]*ad2-A[0,3]*ad3;
end;

function determinante (var A:tmatriz5x5): real;
var
  subA: tmatriz4x4;
  i, j, k, signo: integer;
  acumulador,ad0,ad1,ad2,ad3,ad4: real;

begin
  acumulador:=0;
  signo:=1;
  {ad0}
  subA[0,0]:=A[1,1];
  subA[0,1]:=A[1,2];
  subA[0,2]:=A[1,3];
  subA[0,3]:=A[1,4];
  subA[1,0]:=A[2,1];
  subA[1,1]:=A[2,2];
  subA[1,2]:=A[2,3];
  subA[1,3]:=A[2,4];
  subA[2,0]:=A[3,1];
  subA[2,1]:=A[3,2];
  subA[2,2]:=A[3,3];
```

```
subA[2,3]:=A[3,4];
subA[3,0]:=A[4,1];
subA[3,1]:=A[4,2];
subA[3,2]:=A[4,3];
subA[3,3]:=A[4,4];
ad0:=determinante(subA);

{ad1}
subA[0,0]:=A[1,0];
subA[0,1]:=A[1,2];
subA[0,2]:=A[1,3];
subA[0,3]:=A[1,4];
subA[1,0]:=A[2,0];
subA[1,1]:=A[2,2];
subA[1,2]:=A[2,3];
subA[1,3]:=A[2,4];
subA[2,0]:=A[3,0];
subA[2,1]:=A[3,2];
subA[2,2]:=A[3,3];
subA[2,3]:=A[3,4];
subA[3,0]:=A[4,0];
subA[3,1]:=A[4,2];
subA[3,2]:=A[4,3];
subA[3,3]:=A[4,4];
ad1:=determinante(subA);

{ad2}
subA[0,0]:=A[1,0];
subA[0,1]:=A[1,1];
subA[0,2]:=A[1,3];
subA[0,3]:=A[1,4];
subA[1,0]:=A[2,0];
subA[1,1]:=A[2,1];
subA[1,2]:=A[2,3];
subA[1,3]:=A[2,4];
subA[2,0]:=A[3,0];
subA[2,1]:=A[3,1];
subA[2,2]:=A[3,3];
subA[2,3]:=A[3,4];
subA[3,0]:=A[4,0];
```

```
  subA[3,1]:=A[4,1];
  subA[3,2]:=A[4,3];
  subA[3,3]:=A[4,4];
  ad2:=determinante(subA);

    {ad3}
  subA[0,0]:=A[1,0];
  subA[0,1]:=A[1,1];
  subA[0,2]:=A[1,2];
  subA[0,3]:=A[1,4];
  subA[1,0]:=A[2,0];
  subA[1,1]:=A[2,1];
  subA[1,2]:=A[2,2];
  subA[1,3]:=A[2,4];
  subA[2,0]:=A[3,0];
  subA[2,1]:=A[3,1];
  subA[2,2]:=A[3,2];
  subA[2,3]:=A[3,4];
  subA[3,0]:=A[4,0];
  subA[3,1]:=A[4,1];
  subA[3,2]:=A[4,2];
  subA[3,3]:=A[4,4];
  ad3:=determinante(subA);

  {ad4}
subA[0,0]:=A[1,0];
subA[0,1]:=A[1,1];
subA[0,2]:=A[1,2];
subA[0,3]:=A[1,3];
subA[1,0]:=A[2,0];
subA[1,1]:=A[2,1];
subA[1,2]:=A[2,2];
subA[1,3]:=A[2,3];
subA[2,0]:=A[3,0];
subA[2,1]:=A[3,1];
subA[2,2]:=A[3,2];
subA[2,3]:=A[3,3];
subA[3,0]:=A[4,0];
subA[3,1]:=A[4,1];
subA[3,2]:=A[4,2];
```

```
subA[3,3]:=A[4,3];
ad4:=determinante(subA);


  determinante:=A[0,0]*ad0-A[0,1]*ad1+A[0,2]*ad2-A[0,3]*ad3 + A[0,4]*ad4;
end;


function norma(var b:tvector4):real;
begin
  norma:=sqrt(sqr(b[0])+sqr(b[1])+sqr(b[2])+sqr(b[3]));
end;

function norma(var b:tvector3):real;
begin
  norma:=sqrt(sqr(b[0])+sqr(b[1])+sqr(b[2]));
end;

function norma(var b:tvector5):real;
begin
  norma:=sqrt(sqr(b[0])+sqr(b[1])+sqr(b[2])+sqr(b[3])+sqr(b[4]));
end;

procedure resolver (var A:tmatriz2x2; var v:tvector2; var b: tvector2);
begin
  v[0]:=(b[0]*A[1,1]-A[0,1]*b[1])/(A[0,0]*A[1,1]-A[0,1]*A[1,0]);
  v[1]:=(b[1]*A[0,0]-A[1,0]*b[0])/(A[0,0]*A[1,1]-A[0,1]*A[1,0]);

end;

function resolver (var A:tmatriz3x3; var v:tvector3; var b: tvector3):boolean;
var
  det: real;
  A1: tmatriz3x3;
  i,j: integer;
begin
  det:=determinante(A);

  if det = 0 then resolver:=false else
  begin
```

```
  {Copiar la matriz A en A1}

  for i:=0 to 2 do
    for j:= 0 to 2 do
       A1[i,j]:=A[i,j];

{Calcular v[0] por cramer}
  for i:=0 to 2 do
    A1[i,0]:=b[i];

  v[0]:=determinante(A1)/det;

  for i:=0 to 2 do
    A1[i,0]:=A[i,0];

  {Calcular v[1] por cramer}
  for i:=0 to 2 do
    A1[i,1]:=b[i];

  v[1]:=determinante(A1)/det;

  for i:=0 to 2 do
    A1[i,1]:=A[i,1];

  {Calcular v[2] por cramer}
  for i:=0 to 2 do
    A1[i,2]:=b[i];

  v[2]:=determinante(A1)/det;

  for i:=0 to 2 do
    A1[i,2]:=A[i,2];
  resolver:=true;

  end;
end;


function resolver (var A:tmatriz4x4; var v:tvector4; var b: tvector4):boolean;
var
```

```
  det: real;
  A1: tmatriz4x4;
  i,j: integer;
begin
  det:=determinante(A);
  if det=0 then resolver:=false
  else begin

  {Copiar la matriz A en A1}

  for i:=0 to 3 do
    for j:= 0 to 3 do
       A1[i,j]:=A[i,j];

{Calcular v[0] por cramer}
  for i:=0 to 3 do
    A1[i,0]:=b[i];

  v[0]:=determinante(A1)/det;

  for i:=0 to 3 do
    A1[i,0]:=A[i,0];

  {Calcular v[1] por cramer}
  for i:=0 to 3 do
    A1[i,1]:=b[i];

  v[1]:=determinante(A1)/det;

  for i:=0 to 3 do
    A1[i,1]:=A[i,1];

  {Calcular v[2] por cramer}
  for i:=0 to 3 do
    A1[i,2]:=b[i];

  v[2]:=determinante(A1)/det;

  for i:=0 to 3 do
    A1[i,2]:=A[i,2];
```

```
    {Calcular v[3] por cramer}
  for i:=0 to 3 do
   A1[i,3]:=b[i];

  v[3]:=determinante(A1)/det;

  for i:=0 to 3 do
   A1[i,3]:=A[i,3];

  resolver:=true;
  end;

end;

procedure resolver (var A:tmatriz5x5; var v:tvector5; var b: tvector5);
var
  det: real;
  A1: tmatriz5x5;
  i,j: integer;
begin
  det:=determinante(A);

  {Copiar la matriz A en A1}

  for i:=0 to 4 do
    for j:= 0 to 4 do
       A1[i,j]:=A[i,j];

{Calcular v[0] por cramer}
  for i:=0 to 4 do
   A1[i,0]:=b[i];

  v[0]:=determinante(A1)/det;

  for i:=0 to 4 do
   A1[i,0]:=A[i,0];

  {Calcular v[1] por cramer}
  for i:=0 to 4 do
```

```
    A1[i,1]:=b[i];

  v[1]:=determinante(A1)/det;

  for i:=0 to 4 do
    A1[i,1]:=A[i,1];

  {Calcular v[2] por cramer}
  for i:=0 to 4 do
    A1[i,2]:=b[i];

  v[2]:=determinante(A1)/det;

  for i:=0 to 4 do
    A1[i,2]:=A[i,2];

     {Calcular v[3] por cramer}
  for i:=0 to 4 do
    A1[i,3]:=b[i];

  v[3]:=determinante(A1)/det;

  for i:=0 to 4 do
    A1[i,3]:=A[i,3];

        {Calcular v[4] por cramer}
     for i:=0 to 4 do
       A1[i,4]:=b[i];

     v[4]:=determinante(A1)/det;

     for i:=0 to 4 do
       A1[i,4]:=A[i,4];

end;

procedure invertirmatriz(var A: tmatriz3x3; var Ainversa: tmatriz3x3);
var
  columna, solucion: tvector3;
begin
```

```
    columna[0]:=1;
    columna[1]:=0;
    columna[2]:=0;
    resolver(A,solucion,columna);
    Ainversa[0,0]:=solucion[0];
    Ainversa[1,0]:=solucion[1];
    Ainversa[2,0]:=solucion[2];

    columna[0]:=0;
    columna[1]:=1;
    columna[2]:=0;
    resolver(A,solucion,columna);
    Ainversa[0,1]:=solucion[0];
    Ainversa[1,1]:=solucion[1];
    Ainversa[2,1]:=solucion[2];

    columna[0]:=0;
    columna[1]:=0;
    columna[2]:=1;
    resolver(A,solucion,columna);
    Ainversa[0,2]:=solucion[0];
    Ainversa[1,2]:=solucion[1];
    Ainversa[2,2]:=solucion[2];

end;

procedure esferac (var A:tmatriz4x3; var centro:tvector3; var radio: real);
var
  B: tmatriz3x3;
  indep: tvector3;
  i,j: integer;
begin
  B[0,0]:=2*(A[1,0]-A[0,0]);
  B[0,1]:=2*(A[1,1]-A[0,1]);
  B[0,2]:=2*(A[1,2]-A[0,2]);

  B[1,0]:=2*(A[2,0]-A[0,0]);
  B[1,1]:=2*(A[2,1]-A[0,1]);
  B[1,2]:=2*(A[2,2]-A[0,2]);
```

```
  B[2,0]:=2*(A[3,0]-A[0,0]);
  B[2,1]:=2*(A[3,1]-A[0,1]);
  B[2,2]:=2*(A[3,2]-A[0,2]);

  for i:=0 to 2 do
    begin
    indep[i]:=0;
    for j:=0 to 2 do
      indep[i]:= indep[i]+sqr(A[i+1,j])-sqr(A[0,j]);
    end;

  resolver(B,centro,indep);
  radio:=sqrt(sqr(A[0,0]-centro[0]) +
              sqr(A[0,1]-centro[1]) +
              sqr(A[0,2]-centro[2]) );

end;

procedure cilindroc (var A:tmatriz3x3; var centro:tvector2; var radio: real);
var
  B: tmatriz2x2;
  indep: tvector2;
  i,j: integer;
begin
  B[0,0]:=2*(A[1,0]-A[0,0]);
  B[0,1]:=2*(A[1,1]-A[0,1]);

  B[1,0]:=2*(A[2,0]-A[0,0]);
  B[1,1]:=2*(A[2,1]-A[0,1]);


  for i:=0 to 1 do
    begin
    indep[i]:=0;
    for j:=0 to 1 do
      indep[i]:= indep[i]+sqr(A[i+1,j])-sqr(A[0,j]);
    end;

  resolver(B,centro,indep);
  radio:=sqrt(sqr(A[0,0]-centro[0]) +
```

```
            sqr(A[0,1]-centro[1]) );

end;

procedure fun (var P:tmatriz20x3; npuntos: integer ; var sol:tvector4; var funcion: tvector4);
var
  i,j: integer;
  raiz: real;

begin
     {Inicializamos a cero}
  for i:=0 to 3 do
   funcion[i]:=0;
  {Cargamos los valores iniciales en funcion}
  for i:=0 to npuntos-1 do
   begin
   raiz:=sqrt(sqr(P[i,0] - sol[0]) + sqr(P[i,1] - sol[1]) + sqr(P[i,2] - sol[2]));
   for j:=0 to 2 do
    begin
      funcion[j]:= funcion[j] + 2*sol[j] - (sol[3]*(-2*P[i,j]+2*sol[j]))/raiz - 2 * P[i,j];
    end;
   funcion[3]:= funcion[3] + 2*sol[3] - 2*raiz;
   end;

end;

procedure jac (var P:tmatriz20x3; npuntos: integer ; var sol:tvector4; var jacobiano:tmatriz4x4);
var
  i,j,k:integer;
  raiz,raizCubo:real;
begin
  {Inicializamos a cero}
  for i:=0 to 3 do
   for j:=0 to 3 do
     jacobiano[i,j]:=0;
  {Cargamos los valores iniciales en jacobiano}
  for i:=0 to npuntos-1 do
  begin
  raiz:=sqrt(sqr(P[i,0] - sol[0]) + sqr(P[i,1] - sol[1]) + sqr(P[i,2] - sol[2]));
  raizCubo:=raiz*raiz*raiz;
```

```
    for j:=0 to 2 do
     begin
       for k:=0 to 2 do
       begin
         if j = k then jacobiano[j,k]:= jacobiano[j,k] + 2 +
         0.5 * (sol[3]*(-2*P[i,j]+2*sol[j])*(-2*P[i,j]+2*sol[j]))/raizCubo -
         2*sol[3]/raiz
         else jacobiano[j,k]:= jacobiano[j,k] + 0.5*(sol[3]*(-2*P[i,j]+2*sol[j])*(-2*P[i,k]+2*sol[k]))/raizCubo;
       end;
       jacobiano[j,3]:=jacobiano[j,3] - (-2*P[i,j]+2*sol[j])/raiz;
     end;
    for k:=0 to 2 do
      jacobiano[3,k]:=jacobiano[3,k] - (-2*P[i,k]+2*sol[k])/raiz;
  end;
  jacobiano[3,3]:= 2* npuntos;

end;


{P: matriz con un maximo de 20 puntos
 NPuntos: numero de puntos que hay realmente
 sol: vector con las coordenadas del centro y el radio de la esfera
     Tiene que tener un valor inicial proximo al esperado
}
function NResferac (var P:tmatriz20x3; npuntos: integer ; var sol:tvector4):integer;

var
  jacobiano: tmatriz4x4;
  funcion, y: tvector4;
  raiz,raizCubo: real;
  i,j,k: integer;
  repeticiones: integer;

begin
  repeticiones:=0;

  repeat

 fun(P, npuntos, sol, funcion);
 jac(P, npuntos, sol, jacobiano);
```

```
    resolver(jacobiano,y,funcion);

    for i:=0 to 3 do
      sol[i]:=sol[i]-y[i];


  repeticiones:=repeticiones + 1;
  until (repeticiones = 100) or (norma(y)<0.0001) ; {ejecuta 20 pasos del algoritmo NR o tolerancia pequeña}

  NResferac:=repeticiones;

end;

function Mcilindroc (var P:tmatriz20x3; npuntos: integer ; var sol:tvector3):real;
var
  centro, centroAc: tvector2;
  centro3, punto: tvector3;
  radio, radioAc, radio2: real;
  i,j,k,m,n: integer;
  A: tmatriz3x3;
begin
  centroAc[0]:=0;
  centroAc[1]:=0;
  radioAc:=0;
  n:=0;
  for i:=0 to (npuntos-1) do
     for j:=i+1 to (npuntos-1) do
        for k:=j+1 to (npuntos-1) do
        begin
          for m:=0 to 2 do
            begin
            A[0,m]:=P[i,m];
            A[1,m]:=P[j,m];
            A[2,m]:=P[k,m];
            end;

          cilindroc(A,centro,radio);
          centroAc[0]:=centroAc[0]+centro[0];
          centroAc[1]:=centroAc[1]+centro[1];
          radioAc:=radioAc+radio;
```

```
            n:=n+1;

          end;

  centroAc[0]:=centroAc[0]/n;
  centroAc[1]:=centroAc[1]/n;
  radioAc:=radioAc/n;

  sol[0]:=centroAc[0];
  sol[1]:=centroAc[1];
  sol[2]:=radioAc;

  {calculamos como radio la media de la distancia del centro a los puntos}
  radio2:=0;
  centro3[0]:=centroAc[0];
  centro3[1]:=centroAc[1];
  centro3[2]:=0;
  for i:=0 to (npuntos -1) do
    begin
    punto[0]:=P[i,0];
    punto[1]:=P[i,1];
    punto[2]:=0;
    radio2:=radio2+distanciapuntos(centro3, punto);


    end;

  radio2:=radio2/npuntos;
  Mcilindroc:=radio2;

end;

{P: matriz con un maximo de 20 puntos
 NPuntos: numero de puntos que hay realmente
 sol: vector con las coordenadas del centro y el radio del cilindro
     Tiene que tener un valor inicial proximo al esperado
}
function NRcilindroc (var P:tmatriz20x3; npuntos: integer ; var sol:tvector3):integer;

var
```

```
  jacobiano: tmatriz3x3;
  funcion, y: tvector3;
  raiz,raizCubo: real;
  i,j,k: integer;
  repeticiones: integer;

begin

  repeticiones:=0;
  repeat

{   ShowMessage('npuntos '+ IntToStr(npuntos)+ #10+#13+
  'sol0'+FloatTostr(sol[0]) + #10+#13+
  'sol1'+FloatTostr(sol[1]) + #10+#13+
  'sol2'+FloatTostr(sol[2]) + #10+#13+
  'funcion0'+FloatTostr(funcion[0]) + #10+#13+
  'funcion1'+FloatTostr(funcion[1]) + #10+#13+
  'funcion2'+FloatTostr(funcion[2]) + #10+#13+
  'Matriz P'+#10+#13);    }

 funCilindro(P, npuntos, sol, funcion);
 jacCilindro(P, npuntos, sol, jacobiano);
 resolver(jacobiano,y,funcion);

    for i:=0 to 2 do
      sol[i]:=sol[i]-y[i];


  repeticiones:=repeticiones + 1;
  until (repeticiones = 100000) or (norma(y)<0.001) ; {ejecuta 100 pasos del algoritmo NR o tolerancia pequeña}

  NRcilindroc:=repeticiones;

end;


procedure funCilindro (var P:tmatriz20x3; npuntos: integer ; var sol:tvector3; var funcion: tvector3);
var
  i,j: integer;
  raiz: real;
```

```
begin



  {Inicializamos a cero}
  for i:=0 to 2 do
   funcion[i]:=0;
  {Cargamos los valores iniciales en funcion}
  for i:=0 to npuntos-1 do
   begin
   raiz:=sqrt(sqr(P[i,0] - sol[0]) + sqr(P[i,1] - sol[1]) );
   for j:=0 to 1 do
    begin
     funcion[j]:= funcion[j] + 2*sol[j] - (sol[2]*(-2*P[i,j]+2*sol[j]))/raiz - 2 * P[i,j];
{      Showmessage('Iteracion i: ' + IntTostr(i) +#10+#13+
              'Iteracion j: ' + IntTostr(j) +#10+#13+
              'Funcion j: ' + FloatTostr(funcion[j]) +#10+#13+
              'Sol j: ' + FloatTostr(sol[j]) +#10+#13+
              'sol 3: ' + FloatTostr(sol[3]) +#10+#13+
              'P i j: ' + FloatTostr(P[i,j]) +#10+#13+
              'raiz: ' + FloatTostr(raiz) +#10+#13
              );  }
    end;
   funcion[2]:= funcion[2] + 2*sol[2] - 2*raiz;
   end;

 {  ShowMessage('Dentro de funcilindro'+#10+#13+
  'npuntos '+ IntToStr(npuntos)+ #10+#13+
  'sol0'+FloatTostr(sol[0]) + #10+#13+
  'sol1'+FloatTostr(sol[1]) + #10+#13+
  'sol2'+FloatTostr(sol[2]) + #10+#13+
  'funcion0'+FloatTostr(funcion[0]) + #10+#13+
  'funcion1'+FloatTostr(funcion[1]) + #10+#13+
  'funcion2'+FloatTostr(funcion[2]) + #10+#13+
  'Matriz P'+#10+#13);   }
end;

procedure jacCilindro (var P:tmatriz20x3; npuntos: integer ; var sol:tvector3; var jacobiano:tmatriz3x3);
var
```

```
  i,j,k:integer;
  raiz,raizCubo:real;
begin
  {Inicializamos a cero}
  for i:=0 to 2 do
    for j:=0 to 2 do
      jacobiano[i,j]:=0;
  {Cargamos los valores iniciales en jacobiano}
  for i:=0 to npuntos-1 do
  begin
  raiz:=sqrt(sqr(P[i,0] - sol[0]) + sqr(P[i,1] - sol[1]) );
  raizCubo:=raiz*raiz*raiz;
    for j:=0 to 1 do
      begin
        for k:=0 to 1 do
        begin
          if j = k then jacobiano[j,k]:= jacobiano[j,k] + 2 +
          0.5 * (sol[2]*(-2*P[i,j]+2*sol[j])*(-2*P[i,j]+2*sol[j]))/raizCubo -
          2*sol[2]/raiz
          else jacobiano[j,k]:= jacobiano[j,k] + 0.5*(sol[2]*(-2*P[i,j]+2*sol[j])*(-2*P[i,k]+2*sol[k]))/raizCubo;
        end;
        jacobiano[j,2]:=jacobiano[j,2] - (-2*P[i,j]+2*sol[j])/raiz;
      end;
    for k:=0 to 1 do
      jacobiano[2,k]:=jacobiano[2,k] - (-2*P[i,k]+2*sol[k])/raiz;
  end;
  jacobiano[2,2]:= 2* npuntos;

end;




{P: matriz con un maximo de 20 puntos
 NPuntos: numero de puntos que hay realmente
 sol: vector con las coordenadas del centro y el radio del cilindro
     Tiene que tener un valor inicial proximo al esperado
}
function NRcilindroGiradoc (var P:tmatriz20x3; npuntos: integer ; var sol:tvector5):integer;
```

```
var
  jacobiano: tmatriz5x5;
  funcion, y: tvector5;
  raiz,raizCubo: real;
  i,j,k: integer;
  repeticiones: integer;

begin
  repeticiones:=0;

  repeat

 funCilindroGirado(P, npuntos, sol, funcion);
 jacCilindroGirado(P, npuntos, sol, jacobiano);
 resolver(jacobiano,y,funcion);

    for i:=0 to 4 do
      sol[i]:=sol[i]-y[i];


  repeticiones:=repeticiones + 1;
  until (repeticiones = 50) or (norma(y)<0.0001) ; {ejecuta 100 pasos del algoritmo NR o tolerancia pequeña}

  NRcilindroGiradoc:=repeticiones;

end;


procedure funCilindroGirado (var P:tmatriz20x3; npuntos: integer ; var sol:tvector5; var funcion: tvector5);
var
  i,j: integer;
  raiz, a, b, v, w,r, x, y, z, norma2: real;

begin
     {Inicializamos a cero}
  for i:=0 to 4 do
    funcion[i]:=0;
  {Cargamos los valores iniciales en funcion}
  for i:=0 to npuntos-1 do
    begin
```

```
    a:=sol[0];
    b:=sol[1];
    v:=sol[2];
    w:=sol[3];
    r:=sol[4];
    x:=P[i,0];
    y:=P[i,1];
    z:=P[i,2];

    raiz:=sqrt((sqr(P[i,1] - sol[1] - sol[3]*P[i,2]) + sqr(P[i,0] - sol[0] - sol[2]*P[i,2])
            + sqr((P[i,0] - sol[0])*sol[3] - (P[i,1]-sol[1])*sol[2]))/(sqr(sol[2])+sqr(sol[3])+1));
    norma2:= sqr(sol[2])+sqr(sol[3])+1;

    funcion[0]:= funcion[0] + (raiz - r)*(-2*x+2*a+2*v*z - 2* ((x-a) * w - (y - b)*v)*w)/(raiz*norma2);
    funcion[1]:= funcion[1] + (raiz - r)*(-2*y+2*b+2*w*z + 2* ((x-a) * w - (y - b)*v)*v)/(raiz*norma2);
    funcion[2]:= funcion[2] + ((raiz - r)/raiz)*((-2*(x-a-v*z)*z+2*((x-a)*w-(y-b)*v)*(-y+b))/(norma2)-
                    (2*(sqr(y-b-w*z)+sqr(x-a-v*z)+sqr((x-a)*w-(y-b)*v))*v)/sqr(norma2));
    funcion[3]:= funcion[3] + ((raiz - r)/raiz)*((-2*(y-b-w*z)*z+2*((x-a)*w-(y-b)*v)*(x-a))/(norma2)-
                    (2*(sqr(y-b-w*z)+sqr(x-a-v*z)+sqr((x-a)*w-(y-b)*v))*w)/sqr(norma2));
    funcion[4]:= funcion[4]-2*raiz+2*r;
    end;

end;

procedure jacCilindroGirado (var P:tmatriz20x3; npuntos: integer ; var sol:tvector5; var jacobiano:tmatriz5x5);
var
  i,j,k:integer;
  raiz,raizCubo,a, b, v, w, r, x, y, z, norma2, norma2cubo:real;
begin
  {Inicializamos a cero}
  for i:=0 to 4 do
    for j:=0 to 4 do
      jacobiano[i,j]:=0;
  {Cargamos los valores iniciales en jacobiano}
  for i:=0 to npuntos-1 do
  begin
      a:=sol[0];
    b:=sol[1];
    v:=sol[2];
    w:=sol[3];
```

```
    r:=sol[4];
    x:=P[i,0];
    y:=P[i,1];
    z:=P[i,2];
      raiz:=sqrt((sqr(P[i,1] - sol[1] - sol[3]*P[i,2]) + sqr(P[i,0] - sol[0] - sol[2]*P[i,2])
            + sqr((P[i,0] - sol[0])*sol[3] - (P[i,1]-sol[1])*sol[2]))/(sqr(sol[2])+sqr(sol[3])+1));
  norma2:= sqr(sol[2])+sqr(sol[3])+1;
  norma2cubo:= norma2*norma2*norma2;
  end;
  raizCubo:=raiz*raiz*raiz;
  jacobiano[0,0] :=jacobiano[0,0] + 1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) / (v * v + w
* w + 1) * sqr(-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) / 2 - (raiz - r) / RaizCubo* sqr(-2 * x + 2 * a
+ 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) /sqr(v * v + w * w + 1) / 2 + (raiz - r) /raiz * (2 + 2 * w * w) / (v * v + w
* w + 1);

  jacobiano[0,1] := jacobiano[0,1] +1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) / (v * v + w
* w + 1) * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) * (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w -
(y - b) * v) * w) / 2 - (raiz - r) / RaizCubo* (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) /sqr(v * v + w
* w + 1) * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / 2 - 2 * (raiz - r) /raiz * w * v / (v * v + w * w
+ 1);

  jacobiano[0,2] := jacobiano[0,2] + 1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * ((-2 * (x -
a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) +
sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) * (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) *
w) / 2 - (raiz - r) / RaizCubo* (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) / (v * v + w * w + 1) * ((-2 *
(x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v *
z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) / 2 + (raiz - r) /raiz * (2 * z - 2 * (-y + b) * w) / (v * v + w
* w + 1) - 2 * (raiz - r) /raiz * (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) /sqr(v * v + w * w + 1) * v;

  jacobiano[0,3] := jacobiano[0,3] +1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * ((-2 * (y -
b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) +
sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) * (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w)
/ 2 - (raiz - r) / RaizCubo* (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) / (v * v + w * w + 1) * ((-2 * (y
- b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) +
sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) / 2 + (raiz - r) /raiz * (-0.4e1 * (x - a) * w + 2 * (y - b) * v) / (v
* v + w * w + 1) - 2 * (raiz - r) /raiz * (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) /sqr(v * v + w * w +
1) * w;

  jacobiano[0,4] :=jacobiano[0,4] -1/raiz* (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) / (v * v + w * w
+ 1);
```

```
jacobiano[1,0] := jacobiano[1,0] + 1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) / (v * v + w * w + 1) * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) * (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) / 2 - (raiz - r) / RaizCubo* (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) /sqr(v * v + w * w + 1) * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / 2 - 2 * (raiz - r) /raiz * w * v / (v * v + w * w + 1);

jacobiano[1,1] := jacobiano[1,1] +1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) / (v * v + w * w + 1) * sqr(-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / 2 - (raiz - r) / RaizCubo* sqr(-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) /sqr(v * v + w * w + 1) / 2 + (raiz - r) /raiz * (2 + 2 * v * v) / (v * v + w * w + 1);

jacobiano[1,2] := jacobiano[1,2] +1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / 2 - (raiz - r) / RaizCubo* (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / (v * v + w * w + 1) * ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) / 2 + (raiz - r) /raiz * (2 * (-y + b) * v + 2 * (x - a) * w - 2 * (y - b) * v) / (v * v + w * w + 1) - 2 * (raiz - r) /raiz * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) /sqr(v * v + w * w + 1) * v;

jacobiano[1,3] := jacobiano[1,3] +1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / 2 - (raiz - r) / RaizCubo* (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / (v * v + w * w + 1) * ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) / 2 + (raiz - r) /raiz * (2 * z + 2 * (x - a) * v) / (v * v + w * w + 1) - 2 * (raiz - r) /raiz * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) /sqr(v * v + w * w + 1) * w;

jacobiano[1,4] := jacobiano[1,4] -1/raiz* (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / (v * v + w * w + 1);

jacobiano[2,0] := jacobiano[2,0] + 1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) * (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) / 2 - (raiz - r) / RaizCubo* (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) / (v * v + w * w + 1) * ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) / 2 + (raiz - r) /raiz * ((2 * z - 2 * (-y + b) * w) / (v * v + w * w + 1) - 2 * (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) /sqr(v * v + w * w + 1) * v);

jacobiano[2,1] := jacobiano[2,1] +1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * ((-2 * (x -
```

```
a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / 2 - (raiz - r) / RaizCubo* (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / (v * v + w * w + 1) * ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) / 2 + (raiz - r) /raiz * ((2 * (-y + b) * v + 2 * (x - a) * w - 2 * (y - b) * v) / (v * v + w * w + 1) - 2 * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) /sqr(v * v + w * w + 1) * v);

  jacobiano[2,2] := jacobiano[2,2] + 1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * (v * v + w * w + 1) * sqr((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) / 2 - (raiz - r) / RaizCubo* sqr((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) / 2 + (raiz - r) /raiz * ((2 * z * z + 2 * sqr(-y + b)) / (v * v + w * w + 1) - 0.4e1 * (-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) /sqr(v * v + w * w + 1) * v + 0.8e1 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) / norma2cubo * v * v - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1));

  jacobiano[2,3] := jacobiano[2,3] + 1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * (v * v + w * w + 1) * ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) * ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) / 2 - (raiz - r) / RaizCubo* ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) * ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) / 2 + (raiz - r) /raiz * (2 * (x - a) * (-y + b) / (v * v + w * w + 1) - 2 * (-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) /sqr(v * v + w * w + 1) * w - 2 * (-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) /sqr(v * v + w * w + 1) * v + 0.8e1 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) / norma2cubo * v * w);

  jacobiano[2,4] := jacobiano[2,4] -1/raiz* ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v);

  jacobiano[3,0] := jacobiano[3,0] +1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) * (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) / 2 - (raiz - r) / RaizCubo* (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) / (v * v + w * w + 1) * ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) / 2 + (raiz - r) /raiz * ((-0.4e1 * (x - a) * w + 2 * (y - b) * v) / (v * v + w * w + 1) - 2 * (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) /sqr(v * v + w * w + 1) * w);
```

```
jacobiano[3,1] := jacobiano[3,1] +1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / 2 - (raiz - r) / RaizCubo* (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / (v * v + w * w + 1) * ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) / 2 + (raiz - r) /raiz * ((2 * z + 2 * (x - a) * v) / (v * v + w * w + 1) - 2 * (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) /sqr(v * v + w * w + 1) * w);

jacobiano[3,2] := jacobiano[3,2] +1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * (v * v + w * w + 1) * ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) * ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) / 2 - (raiz - r) / RaizCubo* ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v) * ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) / 2 + (raiz - r) /raiz * (2 * (x - a) * (-y + b) / (v * v + w * w + 1) - 2 * (-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) /sqr(v * v + w * w + 1) * w - 2 * (-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) /sqr(v * v + w * w + 1) * v + 0.8e1 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) / norma2cubo * v * w);

jacobiano[3,3] := jacobiano[3,3] + 1 / (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) * (v * v + w * w + 1) * sqr((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) / 2 - (raiz - r) / RaizCubo* sqr((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w) / 2 + (raiz - r) /raiz * ((2 * z * z + 2 * sqr(x - a)) / (v * v + w * w + 1) - 0.4e1 * (-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) /sqr(v * v + w * w + 1) * w + 0.8e1 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) / norma2cubo * w * w - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1));

jacobiano[3,4] := jacobiano[3,4]-1/raiz* ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w);

jacobiano[4,0] := jacobiano[4,0]-1/raiz* (-2 * x + 2 * a + 2 * v * z - 2 * ((x - a) * w - (y - b) * v) * w) / (v * v + w * w + 1);

jacobiano[4,1] := jacobiano[4,1]-1/raiz* (-2 * y + 2 * b + 2 * w * z + 2 * ((x - a) * w - (y - b) * v) * v) / (v * v + w * w + 1);

jacobiano[4,2] := jacobiano[4,2]-1/raiz* ((-2 * (x - a - v * z) * z + 2 * ((x - a) * w - (y - b) * v) * (-y + b)) / (v * v + w * w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * v);
```

```
  jacobiano[4,3] := jacobiano[4,3]-1/raiz* ((-2 * (y - b - w * z) * z + 2 * ((x - a) * w - (y - b) * v) * (x - a)) / (v * v + w *
w + 1) - 2 * (sqr(y - b - w * z) + sqr(x - a - v * z) + sqr((x - a) * w - (y - b) * v)) /sqr(v * v + w * w + 1) * w);

  jacobiano[4,4] := jacobiano[4,4] + 2;

end;


function G (n: integer; var X: tmatriz20x3; var w: tvector3; var pc: tvector3; var rsqr: real): real;
var
  Y: tmatriz20x3;
  P,wwt,S,A, Ahat: tmatriz3x3;
  B,C1,C2, diff: tvector3;
  i,j,k,l: integer;
  averageSqrLength: real;
  sqrLength: tvectorn;
  trace, error, term: real;

begin
  averageSqrLength:=0;
  OuterProduct(w,wwt);
  for i:=0 to 2 do
     for j:=0 to 2 do
       if i= j then P[i,j]:= 1 - wwt[i,j]
       else P[i,j]:=- wwt[i,j];

  for i:=0 to 2 do
     S[i,i]:=0;
  S[0,1]:=-w[2];
  S[0,2]:=w[1];
  S[1,0]:=w[2];
  S[1,2]:=-w[0];
  S[2,0]:=-w[1];
  S[2,1]:=w[0];

  for i:=0 to 2 do
     for j:=0 to 2 do
        A[i,j]:=0;
```

```
for i:=0 to 2 do
   B[i]:=0;

for i:=0 to (n-1) do
begin
    for j:=0 to 2 do
       c1[j]:=X[i,j];

    MultMatrizVector(P,C1,C2);

    for j:=0 to 2 do
       Y[i,j]:=C2[j];

    sqrLength[i]:=Dot(Y[i],Y[i]);

    OuterProduct(Y[i],wwt);
    sumMatrices(A,wwt,A);

    for j:=0 to 2 do
       b[j]:=b[j]+sqrLength[i]*Y[i,j];

    averageSqrLength:=averageSqrLength+sqrLength[i];

end;

for i:=0 to 2 do
   for j:=0 to 2 do
      A[i,j]:=A[i,j]/n;

for i:=0 to 2 do
   B[i]:=B[i]/n;

averageSqrLength := averageSqrLength /n;
{Ahat = - S*A*S}
for i:=0 to 2 do
   for l:=0 to 2 do
      begin
      Ahat[i,l]:=0;
      for j:=0 to 2 do
         for k:=0 to 2 do
```

```
                Ahat[i,l]:= Ahat[i,l]-S[i,j]*A[j,k]*S[k,l];

          end;

  {Traza de Ahat*A}
  trace:=0;
  for i:=0 to 2 do
    for j:=0 to 2 do
      for k:=0 to 2 do
        trace:=trace+Ahat[i,j]*A[j,k];

  {PC=Ahat*B/Trace}
  MultMatrizVector(Ahat,b,PC);
  for i:=0 to 2 do
    PC[i]:=PC[i]/trace;

  error:=0;
  rsqr:=0;
  for i:=0 to (n-1) do
    begin
    term := sqrLength[i]- averageSqrLength - 2 *Dot(Y[i],PC);
    error:= error + term*term;
    for j:=0 to 2 do
      diff[j]:=PC[j]-Y[i,j];

    rsqr:=rsqr+Dot(diff,diff);
    end;

  error:=error/n;
  rsqr:= rsqr/n;
  G:=error;




end;


procedure OuterProduct(var w: tvector3; var wwt: tmatriz3x3);
var
```

```
  i,j: integer;
begin
  for i:=0 to 2 do
     for j:=0 to 2 do
        wwt[i,j]:=w[i]*w[j];
end;



procedure SumMatrices(var A: tmatriz3x3; var B: tmatriz3x3; var C: tmatriz3x3);
var
 i,j: integer;
begin
  for i:=0 to 2 do
     for j:=0 to 2 do
        C[i,j]:=A[i,j]+B[i,j];
end;

procedure AsigMatrices(var A: tmatriz3x3; var B: tmatriz3x3);
var
 i,j: integer;
begin
  for i:=0 to 2 do
     for j:=0 to 2 do
        A[i,j]:=B[i,j];
end;
procedure MultMatrizVector(var A: tmatriz3x3; var b: tvector3; var c: tvector3);
var
  i: integer;
begin
  for i:=0 to 2 do
     c[i]:=a[i,0]*b[0]+a[i,1]*b[1] + a[i,2]*b[2];
end;

procedure MultVectorMatriz(var b: tvector3; var A: tmatriz3x3;  var c: tvector3);
var
  i: integer;
begin
  for i:=0 to 2 do
     c[i]:=a[0,i]*b[0]+a[1,i]*b[1] + a[2,i]*b[2];
```

```
end;

function Dot(var A: tvector3; var B: tvector3): real;
begin
  Dot:=A[0]*B[0]+A[1]*B[1]+A[2]*B[2];
end;

function FitCylinder (n: integer; var X: tmatriz20x3; var rsqr: real; var C: tvector3; var W: tvector3): real;
var
  average, currentW, currentC: tvector3;
  error, minerror, phi, csphi, snphi, infinity, theta, cstheta, sntheta, currentrsqr: real;
  i,j, imax, jmax,k: integer;

begin
  imax:=640;
  jmax:=640;
  infinity:=100000;

  for i:=0 to 2 do
    average[i]:=0;

  for i:=0 to (n-1) do
    for j:=0 to 2 do
      average[j]:=average[j]+X[i,j];

  for j:=0 to 2 do
    average[j]:=average[j]/n;

  for i:=0 to (n-1) do
    for j:=0 to 2 do
      X[i,j]:=X[i,j]-average[j];

  minerror:=infinity;

  for i:=0 to 2 do
    begin
    w[i]:=0;
    C[i]:=0;
    end;
```

```
  rsqr:=0;
  for j:=0 to jmax do
    begin
    phi:=PI/2 * j /jmax;
    csphi:=cos(phi);
    snphi:=sin(phi);
    for i:=0 to (imax -1) do
      begin
      theta := 2 * PI * i / imax;
      cstheta:= cos(theta);
      sntheta:=sin(theta);
      currentW[0]:=cstheta*snphi;
      currentW[1]:=sntheta*snphi;
      currentW[2]:=csphi;
      error:=G(n,X,currentW,currentC,currentRsqr);
      if error < minerror then
      begin
        minerror:=error;
        for k:=0 to 2 do
          begin
          W[k]:=currentW[k];
          C[k]:=currentC[K]
          end;
        rsqr:=currentrsqr;
      end;

    end;


end;
  FitCylinder:=minError;
end;

function resolvercilindrogirado(var B: tmatriz20x3; npuntos: integer; var A: tmatriz3x3; var u1: tvector3; var radio:real;
var c: tvector3): real;
var
  i,j:integer;
  Ainversa: tmatriz3x3;
  solparcial: tvector3;
```

```
  Anueva: tmatriz3x3;
  centro: tvector2;

  Bi,Bibis,cbis: tvector3;
  error: real;

  {Para mostrar la matriz B}
  cadena:string;

begin

    {Mostramos por pantalla la matriz B}

  {cadena:='';
  for i:=0 to npuntos-1 do
    begin
    for j:=0 to 2 do
    begin
      cadena:=cadena + FloatToStr(B[i,j]);
       cadena:= cadena + '   ';
    end;
    cadena:= cadena + #10 + #13;
    end;

  showmessage(cadena);   }


  {invertir trasponiendo porque es simetrica}
  for i:=0 to 2 do
    for j:=0 to 2 do
      Ainversa[i,j]:=A[j,i];

  {MostrarMatriz3x3(Ainversa); }

  for i:=0 to (npuntos-1) do
    begin
      for j:=0 to 2 do
        bi[j]:=B[i,j];

      multmatrizvector(Ainversa,Bi,BiBis);
```

```
      for j:=0 to 2 do
        B[i,j]:=bibis[j];

    end;

  for i:=0 to 2 do
    for j:=0 to 2 do
      Anueva[i,j]:=B[i,j];

 {Version anterior con los tres primeros puntos  }
{ cilindroc (Anueva, centro, radio);



  c[0]:=centro[0];
  c[1]:=centro[1];
  c[2]:=0; }

  {Mostramos por pantalla la matriz B}

 { cadena:='';
  for i:=0 to npuntos-1 do
    begin
    for j:=0 to 2 do
      begin
      cadena:=cadena + FloatToStr(B[i,j]);
       cadena:= cadena + '   ';
      end;
    cadena:= cadena + #10 + #13;
    end;

  showmessage(cadena);  }


  {Calcular el error y ajustar parametros}
  error:=CalcularCilindro(B,npuntos,solparcial);



  c[0]:=solparcial[0];
  c[1]:=solparcial[1];
```

```
  c[2]:=0;

  cbis[0]:=c[0];
  cbis[1]:=c[1];
  cbis[2]:=c[2];

  radio:=solparcial[2];

  multmatrizvector(A,cbis,c);

  resolvercilindrogirado:=error;


end;

procedure conoc(var P:tmatriz20x3; npuntos: integer; var sol:tvector4);
begin

end;

procedure conoc5puntos(var P:tmatriz5x3; var sol:tvector4);
var
  B: tmatriz4x4;
  indep:tvector4;
  i,j:integer;
begin
  for i:=0 to 3 do
    begin
    B[i,0]:=2*(P[i+1,0]-P[0,0]);
    B[i,1]:=2*(P[i+1,1]-P[0,1]);
    B[i,2]:=(sqr(P[i+1,2])-sqr(P[0,2]));
    B[i,3]:=-2*(P[i+1,2]-P[0,2]);
    indep[i]:=sqr(P[i+1,0]) +sqr(P[i+1,1]) - sqr(P[1,0])- sqr(P[0,1])
    end;
  if determinante(B)=0 then showmessage('Determinante cero')
  else   resolver(B,sol,indep);

end;

procedure funcono (var P:tmatriz4x3; var sol:tvector4; var funcion: tvector4);
```

```
var
  i: integer;
begin
  for i:=0 to 3 do
    funcion[i]:=sqr(P[i,0]-sol[0]) + sqr(P[i,1]-sol[1]) - sol[3] *  sqr(P[i,2]-sol[2]);
end;

procedure jaccono (var P:tmatriz4x3; var sol:tvector4; var jac: tmatriz4x4);
var
  i,j: integer;
begin
  for i:=0 to 3 do
    begin
      jac[i,0]:= -2 * P[i,0] + 2 * sol[0];
      jac[i,1]:= -2 * P[i,1] + 2 * sol[1];
      jac[i,2]:= 2 * sol[3] * (P[i,2] -  sol[2]);
      jac[i,3]:= - sqr( P[i,2] - sol[2]);
    end;

end;

function NRConoc (var P:tmatriz4x3; var sol:tvector4):integer;

var
  jacobiano: tmatriz4x4;
  funcion, y: tvector4;
  raiz,raizCubo: real;
  i,j,k: integer;
  repeticiones: integer;

begin
  repeticiones:=0;

  repeat

 funcono(P, sol, funcion);
 jaccono(P, sol, jacobiano);
    resolver(jacobiano,y,funcion);

    for i:=0 to 3 do
```

```
      sol[i]:=sol[i]-y[i];


   repeticiones:=repeticiones + 1;
   until (repeticiones = 100000) or (norma(y)<0.0001) ; {ejecuta 20 pasos del algoritmo NR o tolerancia pequeña}

   nrconoc:=repeticiones;

end;


{NR cono recto con npuntos. Sobrecarga de los metodos anteriores}

function NRConoc (var P:tmatriz20x3; npuntos: integer; var sol:tvector4):integer;

var
   jacobiano: tmatriz4x4;
   funcion, y: tvector4;
   i,j,k: integer;
   repeticiones: integer;
   texto:string;

begin
   repeticiones:=0;

   repeat
  funcono(P, npuntos, sol, funcion);
  jaccono(P, npuntos, sol, jacobiano);
  if repeticiones=0 then
  begin
    texto:='';
    for i:=0 to 3 do

       begin
         texto:=texto + '   ' +floattostr(funcion[i]);
       end;
         texto:=texto+#10+#13;

    showmessage(texto);
```

```
  end;
    resolver(jacobiano,y,funcion);

    for i:=0 to 3 do
      sol[i]:=sol[i]-y[i];


   repeticiones:=repeticiones + 1;
   until (repeticiones = 100) or (norma(y)<0.0001) ; {ejecuta 20 pasos del algoritmo NR o tolerancia pequeña}

   nrconoc:=repeticiones;

end;

procedure funcono (var P:tmatriz20x3; npuntos: integer; var sol:tvector4; var funcion: tvector4);
var
   i,j: integer;
   x,y,z,a,b,c,d: real;
   raizab, raiz1d, numeradorzc, numzdd, numcdd, cuadradosab, raizcuboab,pargrande, raiz1dcubo, raizd1d: real;

begin
   a:=sol[0];
   b:=sol[1];
   c:=sol[2];
   d:=sol[3];



   for i:=0 to 3 do
     funcion[i]:=0;

   for i:=0 to (npuntos-1) do
     begin
     x:=P[i,0];
     y:=P[i,1];
     z:=P[i,2];
     cuadradosab:=sqr(x-a)+sqr(y-b);
     raizab:=sqrt(cuadradosab);
     raizcuboab:=raizab*raizab*raizab;
     if d<=0 then d:=-d;
```

```
    raiz1d:=sqrt(1+d);
    raizd1d:=sqrt(d/(1+d));
    raiz1dcubo:=raiz1d*raiz1d*raiz1d;
    numeradorzc:=raizab/raiz1d+(c-z)*raizd1d;
    numzdd:=z * (1/(1+d)- d /sqr(1+d));
    numcdd:=c * (1/(1+d)- d /sqr(1+d));
    pargrande:=-0.5*raizab/(raiz1dcubo)-0.5*numzdd/(raizd1d)+0.5*numcdd/raizd1d;

    funcion[0]:=funcion[0]+numeradorzc*(-2 * x + 2* a)/(raizab*raiz1d);
    funcion[1]:=funcion[1]+numeradorzc*(-2 * y + 2* b)/(raizab*raiz1d);
    funcion[2]:=funcion[2]- 2 *numeradorzc*raizd1d;
    funcion[3]:=funcion[3]+2*numeradorzc*pargrande;
    end;

end;

procedure jaccono (var P:tmatriz20x3; npuntos: integer; var sol:tvector4; var jac: tmatriz4x4);
var
  i,j: integer;
     x,y,z,a,b,c,d: real;
  raizab, raiz1d, numeradorzc, numzdd, numcdd, cuadradosab, raizcuboab,pargrande, raiz1dcubo, raizd1d: real;

begin
 a:=sol[0];
  b:=sol[1];
  c:=sol[2];
  d:=sol[3];


  for i:=0 to 3 do
    for j:=0 to 3 do
    jac[i,j]:=0;

  for i:=0 to (npuntos-1) do
    begin
    x:=P[i,0];
    y:=P[i,1];
    z:=P[i,2];
    if d<0 then d:=-d;
    cuadradosab:=sqr(x-a)+sqr(y-b);
```

```
    raizab:=sqrt(cuadradosab);
    raizcuboab:=raizab*raizab*raizab;
    raiz1d:=sqrt(1+d);
    raizd1d:=sqrt(d/(1+d));
    raiz1dcubo:=raiz1d*raiz1d*raiz1d;
    numeradorzc:=raizab/raiz1d+(c-z)*raizd1d;
    numzdd:=z * (1/(1+d)- d /sqr(1+d));
    numcdd:=c * (1/(1+d)- d /sqr(1+d));
    pargrande:=-0.5*raizab/(raiz1dcubo)-0.5*numzdd/(raizd1d)+0.5*numcdd/raizd1d;

          jac[0,0]:=jac[0,0] + 0.5*sqr(-2*x+2*a) / (cuadradosab * (1+d)) - 0.5 *numeradorzc * sqr(-2*x+2*a)/
(raizcuboab*raiz1d)
           +2*numeradorzc/(raizab*raiz1d);
     jac[0,1]:=jac[0,1] + 0.5 * (-2 * y + 2 * b) * (-2 * x + 2 * a) /(cuadradosab*(1+d)) -0.5*numeradorzc*(-2*x+2*a)*(-
2*y+2*b)/
           (raizcuboab*raiz1d);
    jac[0,2]:=jac[0,2]+(-2*x+2*a)*raizd1d/(raizab*raiz1d);
    jac[0,3]:=jac[0,3]+(-2*x+2*a)*pargrande/(raizab*raiz1d)-0.5*numeradorzc*(-2*x+2*a)/(raizab*raiz1dcubo);

    jac[1,1]:=jac[1,1]+0.5*sqr(-2*y+2*b)/(cuadradosab*(1+d))-0.5*numeradorzc*sqr(-2*y+2*b)/(raizcuboab*raiz1d)+
           2*numeradorzc/(raizab*raiz1d);
    jac[1,2]:=jac[1,2]+(-2*y+2*b)*raizd1d/(raizab*raiz1d);
    jac[1,3]:=jac[1,3]+(-2*y+2*b)*pargrande/(raizab*raiz1d)-0.5*numeradorzc*(-2*y+2*b)/(raizab*raiz1dcubo);

    jac[2,2]:=jac[2,2]+2*d/(1+d);
                                                jac[2,3]:=jac[2,3]+2*(-0.5*raizab/raiz1dcubo-0.5*numzdd/(raizd1d)
+0.5*numcdd/raizd1d)*raizd1d+numeradorzc*(1/(1+d)-d/sqr(1+d))/raizd1d;

         jac[3,3]:=jac[3,3]+2*sqr(pargrande)+2*numeradorzc*(0.75*raizab/(sqr(1+d)*raiz1d)+0.25*numzdd*(1/(1+d)- d
/sqr(1+d))/
           ((d/(1+d)) * sqrt(d/(1+d))) -0.5*z*(-2/sqr(1+d)+2*d/(sqr(1+d)*(1+d))) /sqrt(d/(1+d)) -
           0.25*numcdd*(1/(1+d)- d /sqr(1+d))/
           ((d/(1+d)) * sqrt(d/(1+d))) +0.5*c*(-2/sqr(1+d)+2*d/(sqr(1+d)*(1+d))) /sqrt(d/(1+d)));

    end;

  for i:=0 to 3 do
    for j:=i+1 to 3 do
      jac[j,i]:= jac[i,j];
```

```
end;


procedure eliminarcd (var A:tmatriz4x4; var indep: tvector4; var A2:tmatriz3x3; var indep2: tvector3);
var
  i: integer;
begin
 for i:=0 to 2 do
   begin
     A2[i,0]:= A[i+1,0] * A[0,2] - A[0,0]*A[i+1,2];
     A2[i,1]:= A[i+1,1] * A[0,2] - A[0,1]*A[i+1,2];
     A2[i,2]:= A[i+1,3] * A[0,2] - A[0,3]*A[i+1,2];
     indep2[i]:= indep[i+1] * A[0,2] - indep[0]*A[i+1,2];
   end;
end;


procedure resolversistemacono(neq: integer; var A: tmatriz20x4; var sol: tvector4; var indep: tvectorn);
{neq contiene el ultimo indice de la matriz A: numero de ecuaciones menos uno}
var
 B: tmatriz4x4;
 c: tvector4;
 i,j,k: integer;
begin
  {B=A'·A}
 for i:=0 to 3 do
   for j:=0 to 3 do
     B[i,j]:=0;

 for i:=0 to 3 do
   for j:=0 to 3 do
     for k:=0 to neq do
       B[i,j]:=B[i,j]+ A[k,i]*A[k,j];

 {c=A'b}
 for i:=0 to 3 do
   c[i]:=0;

 for i:=0 to 3 do
   for k:=0 to neq do
```

```
      c[i]:=c[i]+A[k,i]*indep[k];

  resolver(B,sol,c);



end;

function CalcularEsfera (var B:tmatriz20x3; npuntos: integer ; var sol:tvector4):integer;
var
  i,j: integer;
  A:tmatriz4x3;
  v:tvector3;
  radio:real;
begin
    {Calculamos primero de forma exacta a partir de 4 puntos}
  for i:=0 to 3 do
    for j:=0 to 2 do
      A[i,j]:= B[i,j];

  esferac(A,v,radio);

  {Tenemos la solucion inicial sol}
  sol[0]:=v[0];
  sol[1]:=v[1];
  sol[2]:=v[2];
  sol[3]:=radio;

  {Newton Raphson con los parametros necesarios}
  CalcularEsfera:=NResferac(B,npuntos, sol);
end;

 {Calcula el cilindro y devuelve el error maximo}
function CalcularCilindro (var B:tmatriz20x3; npuntos: integer ; var sol:tvector3):real;
var
A: tmatriz3x3;
v: tvector2;
puntop, centro: tvector3;
i,j, iteraciones: integer;
radio,error,dist: real;
```

```
begin

  {Calculamos primero de forma exacta a partir de 3 puntos}
  for i:=0 to 2 do
    for j:=0 to 2 do
      A[i,j]:= B[i,j];

  cilindroc(A,v,radio);

  {Tenemos la solucion inicial sol}
  sol[0]:=v[0];
  sol[1]:=v[1];
  sol[2]:=radio;

{  Showmessage('Sol inicial:' + FloatToStr(sol[0]) + #10 + #13
  + FloatToStr(sol[1]) + #10 + #13 + FloatToStr(sol[2])
  + #10 + #13 + 'npuntos: ' + IntToStr(npuntos));  }

  NRcilindroc(B,npuntos, sol);

          {Calcular distancia maxima para tener una estimacion del error}
     error:=0;
     centro[0]:=sol[0];
     centro[1]:=sol[1];
     centro[2]:=0;
     for i:=0 to npuntos-1 do
       begin
          for j:=0 to 1 do
            puntop[j]:=B[i,j];

          puntop[2]:=0;

          dist:= abs(distanciaPuntoPunto(puntop,centro)-sol[2]);
         if dist > error then error := dist;
       end;
       {Fin del calculo del error}
  CalcularCilindro:= error;
end;
```

```
function CalcularCono (var B:tmatriz20x3; npuntos: integer ; var sol:tvector4):integer;
var
  i: integer;
  A: tmatriz20x4;
  indep: tvectorn;
begin
    for i:=0 to npuntos-2 do
    begin
      A[i,0]:= 2 * (B[i+1,0] - B[0,0]);
      A[i,1]:= 2 * (B[i+1,1] - B[0,1]);
      A[i,2]:= - 2 * (B[i+1,2] - B[0,2]);
      A[i,3]:= (B[i+1,2] * B[i+1,2] - B[0,2]*B[0,2]);
      indep[i]:=  B[i+1,0] * B[i+1,0] - B[0,0]*B[0,0] + B[i+1,1] * B[i+1,1] - B[0,1]*B[0,1];
    end;

   resolversistemacono(npuntos-2,A,sol,indep);
   sol[2]:=sol[2]/sol[3];
end;

function distanciaPuntoPunto(var a,b: tvector3):real;
var
  c: tvector3;
  i: integer;
begin
   for i:=0 to 2 do
      c[i]:=a[i]-b[i];
   distanciaPuntoPunto:=norma(c);

end;

function distanciaPuntoPlano(var a,b,n: tvector3):real;
begin
  { showmessage('Punto: ' + #10 +#13 +FloatTostr(a[0])
                    + #10 +#13 +FloatTostr(a[1])
                    + #10 +#13 +FloatTostr(a[2])
             +'Punto' + #10 +#13 +FloatTostr(b[0])
                    + #10 +#13 +FloatTostr(b[1])
                    + #10 +#13 +FloatTostr(b[2])
```

```
            +'Vector' + #10 +#13 +FloatTostr(n[0])
                  + #10 +#13 +FloatTostr(n[1])
                  + #10 +#13 +FloatTostr(n[2])
                  );  }
  distanciaPuntoPlano:=abs((a[0]-b[0])*n[0]+(a[1]-b[1])*n[1]+(a[2]-b[2])*n[2]) /norma(n);

end;

function distanciaPuntoRecta(var a,b,n: tvector3):real;
var
 c: tvector3;

begin
  c[0]:= (a[1]-b[1])* n[2] - (a[2]-b[2])* n[1];
  c[1]:= (-1)*((a[0]-b[0])* n[2] - (a[2]-b[2])* n[0]);
  c[2]:= (a[0]-b[0])* n[1] - (a[1]-b[1])* n[0];

  distanciaPuntoRecta:=norma(c) /norma(n);

end;

function distanciaRectaRecta(var a,b,n,m: tvector3):real;
var
 c: tvector3;

begin
  {Calculamos primero el prod vect. n^m}
  c[0]:= n[1]*m[2]-n[2]*m[1];
  c[1]:= n[2]*m[0]-n[0]*m[2];
  c[2]:= n[0]*m[1]-n[1]*m[0];

  if norma(c) = 0 then distanciaRectaRecta:=0
  else distanciaRectaRecta:=abs((a[0]-b[0])*c[0]+(a[1]-b[1])*c[1]+(a[2]-b[2])*c[2]) /norma(c);

end;

function terminoindep (var B:tmatriz20x3; npuntos: integer ; var sol:tvector3):real;
var
 i,j: integer;
 Ax,By,Cz:real;
```

```
begin
  Ax:=0;
  By:=0;
  Cz:=0;

 for  i:=0 to npuntos-1 do
 begin
   ax:=ax+ B[i,0];
   by:=by+B[i,1] ;
   cz:=cz+B[i,2];
 end;
 terminoindep:=(ax*sol[0]+by*sol[1]+cz*sol[2])/npuntos;
end;

procedure puntodelplano(var P:tvector3; var sol:tvector3; d: real);
begin
 if not (sol[0] = 0) then
 begin
   P[0]:= (d-P[1]*sol[1]-P[2]*sol[2])/sol[0];
 end
 else if not (sol[1] = 0) then
 begin
   P[1]:= (d-P[0]*sol[0]-P[2]*sol[2])/sol[1];
 end
 else
 begin
   P[2]:= (d-P[1]*sol[1]-P[0]*sol[0])/sol[2];
 end
end;


function CalcularConoMedias (var B:tmatriz20x3; npuntos: integer ; var sol:tvector4):real;
var
 temp, puntop, puntoq, vector: tvector3;
 soltemp:tvector4;
 i,j:integer;
 error, dist,x,y: real;
 C:tmatriz20x2;
 cadena: string;
 sumxy, sumx, sumy, sumx2, m,n,d:real;
```

```
  radiocono:real;

begin
{Iniciamos el vector solucion a cero}
  for j:=0 to 3 do
    soltemp[j]:=0;


  {Cambiamos en la matriz el punto i con el punto cero}
  for i:=0 to npuntos-1 do
    begin
    for j:=0 to 2 do
      begin
        temp[j]:=B[i,j];
        B[i,j]:=B[0,j];
        B[0,j]:=temp[j];
      end;
    {Calculamos los parametros del cono y deshacemos el cambio}
    CalcularCono(B, npuntos, sol);
    for j:=0 to 3 do
      soltemp[j]:=soltemp[j]+sol[j];
    for j:=0 to 2 do
      begin
        temp[j]:=B[i,j];
        B[i,j]:=B[0,j];
        B[0,j]:=temp[j];
      end;
    end;

  {hacemos la media de todos los parametros calculados}
  for j:=0 to 3 do
    sol[j]:=soltemp[j]/npuntos;

  {Hasta aqui en sol[0] y sol[1] tenemos la proyeccion del vertice
  sobre el plano z=0}
  {Hacemos una traslacion para llevar el eje del cono al eje z}
    for i:=0 to npuntos-1 do
      for j:=0 to 1 do
        B[i,j]:=B[i,j]-sol[j];
    {Giramos los puntos sobre el eje z para llevarlos al primer cuadrante.
```

```
Lo guardamos en la matriz C}
for i:=0 to npuntos-1 do
  begin
    C[i,0]:=sqrt(B[i,0]*B[i,0]+B[i,1]*B[i,1]);
    C[i,1]:=B[i,2];
  end;
{
{Mostramos la matriz C}
cadena:='';
for i:=0 to npuntos-1 do
  begin
    cadena:=cadena+ FloatTostr(C[i,0]) + '   ' + FloatTostr(C[i,1])
            + #10+#13;
  end;
showmessage(cadena);}

{Calculamos la recta de regresion y=mx+n donde
m=(npuntos*sum xiyi - sum xi * sum yi) /d
n = (sum xi^2 * sum yi - sum xi * sum xiyi) / d
d = npuntos* sum xi^2 - sum x_i * sum x_i}

sumxy:=0;
sumx:=0;
sumy:=0;
sumx2:=0;
m:=0;
n:=0;
d:=0;


for i:=0 to npuntos -1 do
  begin
    sumxy:=sumxy + C[i,0]*C[i,1];
    sumx:= sumx + C[i,0];
    sumy:= sumy + C[i,1];
    sumx2:=sumx2 + C[i,0]*C[i,0];
  end;

d:=npuntos*sumx2-sumx*sumx;
m:=(npuntos*sumxy - sumx *sumy)/d;
```

```
   n:=(sumx2*sumy-sumx*sumxy)/d;

   {La altura del vertice corresponde a x=0 en la recta de regresion}
   sol[2]:=n;

   {El radio del cono corresponde al valor de x haciendo y=0 en la recta}

   radiocono:=abs(-n/m); {El valor absoluto hace que la formula funcione
                          para un cono en cualquier posicion}

   {El parametro d en (x-a)^2 + (y-b)^2 = d(z-c)^2  se obtiene haciendo z= 0
   entonces d * c^2 = radiocono}
   sol[3]:=radiocono*radiocono/(sol[2]*sol[2]);

   {
   {Mostrar por pantalla vertice y radio cono}

   Showmessage('Vertice: '
    +#10 +#13 + Floattostr(sol[0])
    +#10 +#13 + Floattostr(sol[1])
    +#10 +#13 + Floattostr(sol[2])
     +#10 +#13 + Floattostr(radiocono)
    );
    }

  {Calcular distancia maxima para tener una estimacion del error}
error:=0;

for i:=0 to npuntos-1 do
  begin
  dist:=abs(C[i,1]- m * C[i,0]-n);
  if dist > error then error:=dist;
  end;

{
{Eje del cono en puntoq y vector}
puntoq[0]:= sol[0];
puntoq[1]:=sol[1];
puntoq[2]:=0;
```

```
vector[0]:=0;
vector[1]:=0;
vector[2]:=1;

for i:=0 to npuntos-1 do
begin
  for j:=0 to 2 do
    puntop[j]:=B[i,j];

  x:= distanciaPuntoRecta(puntop,puntoq,vector);
  y:=puntop[2];
  if (y <= sol[2]) then dist:=abs(x+sol[3]*y-sol[3]*sol[2])/(sqrt(1+sol[3]))
  else dist:=abs(-x+sol[3]*y-sol[3]*sol[2])/(sqrt(1+sol[3]));
 {  Showmessage('Punto: ' + #10 + #13 + 'a: ' + FloatTostr(puntop[0])
              + #10 + #13 + 'b: '+ FloatTostr(puntop[1])
              + #10 + #13 + 'c: '+ FloatTostr(puntop[2])
              + #10 + #13 + 'distancia: '+ FloatTostr(distanciaPuntoPunto(centro, puntop))); }
  if dist > error then error := dist;
end; }
{Fin del calculo del error}
  CalcularConoMedias:=error;

end;


function CalcularCilindroMedias (var B:tmatriz20x3; npuntos: integer ; var sol:tvector3):real;
var
  temp, puntop, puntoq, vector: tvector3;
  soltemp:tvector3;
  i,j:integer;
  error, dist,x,y: real;

begin
{Iniciamos el vector solucion a cero}
  for j:=0 to 2 do
    soltemp[j]:=0;


  {Cambiamos en la matriz el punto i con el punto cero}
  for i:=0 to npuntos-1 do
```

```
   begin
   for j:=0 to 2 do
     begin
       temp[j]:=B[i,j];
       B[i,j]:=B[0,j];
       B[0,j]:=temp[j];
     end;
   {Calculamos los parametros del cono y deshacemos el cambio}
   CalcularCilindro(B, npuntos, sol);
   for j:=0 to 3 do
     soltemp[j]:=soltemp[j]+sol[j];
   for j:=0 to 2 do
     begin
       temp[j]:=B[i,j];
       B[i,j]:=B[0,j];
       B[0,j]:=temp[j];
     end;
   end;

  {hacemos la media de todos los parametros calculados}
  for j:=0 to 3 do
    sol[j]:=soltemp[j]/npuntos;

  {Calcular distancia maxima para tener una estimacion del error}
error:=0;

{Eje del cono en puntoq y vector}
puntoq[0]:= sol[0];
puntoq[1]:=sol[1];
puntoq[2]:=0;

vector[0]:=0;
vector[1]:=0;
vector[2]:=1;

for i:=0 to npuntos-1 do
begin
   for j:=0 to 2 do
     puntop[j]:=B[i,j];
```

```
  x:= distanciaPuntoRecta(puntop,puntoq,vector);
  y:=puntop[2];
  if (y <= sol[2]) then dist:=abs(x+sol[3]*y-sol[3]*sol[2])/(sqrt(1+sol[3]))
  else dist:=abs(-x+sol[3]*y-sol[3]*sol[2])/(sqrt(1+sol[3]));
 {  Showmessage('Punto: ' + #10 + #13 + 'a: ' + FloatTostr(puntop[0])
              + #10 + #13 + 'b: '+ FloatTostr(puntop[1])
              + #10 + #13 + 'c: '+ FloatTostr(puntop[2])
              + #10 + #13 + 'distancia: '+ FloatTostr(distanciaPuntoPunto(centro, puntop))); }
  if dist > error then error := dist;
end;
{Fin del calculo del error}
  CalcularCilindroMedias:=error;

end;



{Calcula un punto de intersecion P3 dados dos planos}
function puntodosplanos (var P1,v1,P2,v2,P3:tvector3):boolean;
var
 A:Tmatriz2x2;
 sol,indep:tvector2;

begin
 if   (0=v1[1]*v2[2]-v1[2]*v2[1]) and
      (0=v1[2]*v2[0]-v1[0]*v2[2]) and
      (0=v1[0]*v2[1]-v1[1]*v2[0])
 then
   begin
    puntodosplanos:=false;
    showmessage('v ' + FloatTostr(v1[0]) +#10+#13
                + FloatTostr(v1[1]) +#10+#13
                + FloatTostr(v1[2]) +#10+#13
                + FloatTostr(v2[0]) +#10+#13
                + FloatTostr(v2[1]) +#10+#13
                + FloatTostr(v2[2]) +#10+#13 );
   end
 else
 begin
   {   showmessage('dentro');   }
```

```
A[0,0]:=v1[1];
A[0,1]:=v1[2];
A[1,0]:=v2[1];
A[1,1]:=v2[2];
sol[0]:=0;
sol[1]:=0;
indep[0]:=P1[0]*v1[0]+P1[1]*v1[1]+P1[2]*v1[2];
indep[1]:=P2[0]*v2[0]+P2[1]*v2[1]+P2[2]*v2[2];
if not (det2(A) = 0) then
  begin
    resolver(A,sol,indep);
    P3[0]:=0;
    p3[1]:=sol[0];
    p3[2]:=sol[1];
  end
else
  begin
   A[0,0]:=v1[0];
   A[0,1]:=v1[2];
   A[1,0]:=v2[0];
   A[1,1]:=v2[2];
   if not (det2(A) = 0) then
   begin
     resolver(A,sol,indep);
     P3[0]:=sol[0];
     p3[1]:=0;
     p3[2]:=sol[1];
   end
   else
     begin
       A[0,0]:=v1[0];
       A[0,1]:=v1[1];
       A[1,0]:=v2[0];
       A[1,1]:=v2[1];
       if not (det2(A) = 0) then
       begin
         resolver(A,sol,indep);
         P3[0]:=sol[0];
         p3[1]:=sol[1];
         p3[2]:=0;
```

```
            end
            else begin
                showmessage('Error: planos paralelos');
              end;
           end;
        end;

  puntodosplanos:=true;
  end;
end;


function det2 (var A:tmatriz2x2): real;
begin
  det2:=A[0,0]*A[1,1]-A[0,1]*A[1,0];
end;

function arccos(x:real):real;
begin
  if x=0 then arccos:=PI/2 else arccos:= arctan(sqrt(1-x*x)/x);
end;

end.
```